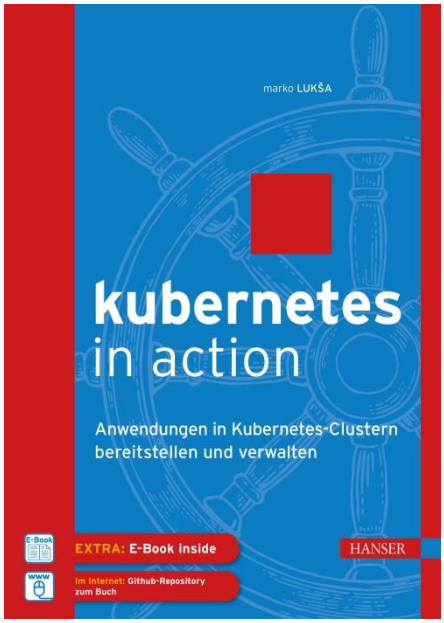
HANSER



Leseprobe

zu

Kubernetes in Action

von Marko Lukša

ISBN (Buch): 978-3-446-45510-8
ISBN (E-Book): 978-3-446-45602-0

Weitere Informationen und Bestellungen unter
https://www.hanser-fachbuch.de/
sowie im Buchhandel

Inhalt

Vorwort ..... **XXI**
Der Autor XXII
Danksagung zur englischsprachigen Ausgabe XXIII

Über dieses Buch ..... **XXV**
Zielgruppe XXV
Der Aufbau dieses Buches XXVI
Der Code XXVII
Das Forum zum Buch XXVIII
Sonstige Onlinequellen XXVIII

Teil I: Überblick **1**

1 Einführung in Kubernetes **3**
1.1 Der Bedarf für ein System wie Kubernetes 4
1.1.1 Von monolithischen Anwendungen zu Microservices 4
1.1.2 Eine konsistente Umgebung für Anwendungen bereitstellen 8
1.1.3 Übergang zu Continuous Delivery: DevOps und NoOps 8
1.2 Containertechnologien 10
1.2.1 Was sind Container? 10
1.2.2 Die Containerplattform Docker 14
1.2.3 Die Docker-Alternative rkt 18
1.3 Kubernetes 19
1.3.1 Die Ursprünge 19
1.3.2 Kubernetes im Überblick 19
1.3.3 Die Architektur eines Kubernetes-Clusters 21
1.3.4 Anwendungen auf Kubernetes ausführen 22
1.3.5 Vorteile der Verwendung von Kubernetes 24
1.4 Zusammenfassung 27

2 Erste Schritte mit Docker und Kubernetes ... 29
2.1 Containerimages mit Docker erstellen, ausführen und teilen ... 29
2.1.1 Docker installieren und einen Hello-world-Container ausführen ... 30
2.1.2 Eine triviale Node.js-Anwendung erstellen ... 32
2.1.3 Eine Docker-Datei für das Image erstellen ... 33
2.1.4 Das Containerimage erstellen ... 33
2.1.5 Das Containerimage ausführen ... 36
2.1.6 Das Innenleben eines laufenden Containers untersuchen ... 37
2.1.7 Container anhalten und entfernen ... 39
2.1.8 Das Image zu einer Registry hochladen ... 39
2.2 Kubernetes-Cluster einrichten ... 41
2.2.1 Einen lokalen Kubernetes-Cluster mit einem Knoten mithilfe von Minikube ausführen ... 41
2.2.2 Gehostete GKE-Cluster ... 43
2.2.3 Einen Alias und die Befehlszeilenvervollständigung für kubectl einrichten ... 46
2.3 Eine erste Anwendung in Kubernetes ausführen ... 47
2.3.1 Die Node.js-Anwendung bereitstellen ... 48
2.3.2 Auf die Webanwendung zugreifen ... 51
2.3.3 Die logischen Bestandteile des Systems ... 52
2.3.4 Anwendungen horizontal skalieren ... 54
2.3.5 Auf welchen Knoten läuft die Anwendung? ... 57
2.3.6 Das Kubernetes-Dashboard ... 58
2.4 Zusammenfassung ... 59

Teil II: Grundlagen ... 61

3 Pods: Container in Kubernetes ausführen ... 63
3.1 Einführung in Pods ... 63
3.1.1 Wozu benötigen wir Pods? ... 64
3.1.2 Grundlagen von Pods ... 65
3.1.3 Container auf Pods verteilen ... 66
3.2 Pods aus YAML- und JSON-Deskriptoren erstellen ... 69
3.2.1 Den YAML-Deskriptor eines bestehenden Pods untersuchen ... 69
3.2.2 Einen einfachen YAML-Deskriptor für einen Pod schreiben ... 71
3.2.3 Einen Pod mit kubectl create erstellen ... 73
3.2.4 Anwendungsprotokolle anzeigen ... 74
3.2.5 Anforderungen an den Pod senden ... 75
3.3 Pods mithilfe von Labels ordnen ... 76
3.3.1 Einführung in Labels ... 77
3.3.2 Labels beim Erstellen eines Pods angeben ... 78
3.3.3 Labels vorhandener Pods ändern ... 79
3.4 Eine Auswahl der Pods mithilfe von Labelselektoren auflisten ... 80
3.4.1 Pods anhand eines Labelselektors auflisten ... 80

3.4.2 Labelselektoren mit mehreren Bedingungen ... 82
3.5 Die Podzuweisung mithilfe von Labels und Selektoren einschränken ... 82
3.5.1 Labels zur Klassifizierung von Arbeitsknoten ... 83
3.5.2 Pods bestimmten Knoten zuweisen ... 84
3.5.3 Zuweisung zu einem einzelnen Knoten ... 84
3.6 Pods mit Anmerkungen versehen ... 85
3.6.1 Die Anmerkungen zu einem Objekt einsehen ... 85
3.6.2 Anmerkungen hinzufügen und ändern ... 86
3.7 Ressourcen mithilfe von Namespaces gruppieren ... 86
3.7.1 Der Bedarf für Namespaces ... 87
3.7.2 Andere Namespaces und die zugehörigen Pods finden ... 87
3.7.3 Namespaces erstellen ... 88
3.7.4 Objekte in anderen Namespaces verwalten ... 89
3.7.5 Die Trennung der Namespaces ... 90
3.8 Pods stoppen und entfernen ... 90
3.8.1 Pods unter Angabe des Namens löschen ... 90
3.8.2 Pods mithilfe von Labelselektoren löschen ... 91
3.8.3 Pods durch Entfernen eines ganzen Namespaces löschen ... 91
3.8.4 Alle Pods in einem Namespace löschen und den Namespace erhalten ... 92
3.8.5 (Fast) alle Ressourcen in einem Namespace löschen ... 92
3.9 Zusammenfassung ... 93

4 Replikationscontroller & Co.: Verwaltete Pods bereitstellen ... 95
4.1 Pods funktionsfähig halten ... 96
4.1.1 Aktivitätssonden ... 96
4.1.2 HTTP-Aktivitätssonden erstellen ... 97
4.1.3 Eine Aktivitätssonde in Aktion ... 98
4.1.4 Weitere Eigenschaften der Aktivitätssonde festlegen ... 99
4.1.5 Wirkungsvolle Aktivitätssonden erstellen ... 100
4.2 Replikationscontroller ... 102
4.2.1 Die Funktionsweise von Replikationscontrollern ... 103
4.2.2 Einen Replikationscontroller erstellen ... 105
4.2.3 Der Replikationscontroller in Aktion ... 106
4.2.4 Pods in den Gültigkeitsbereich eines Replikationscontrollers bringen und daraus entfernen ... 111
4.2.5 Das Pod-Template ändern ... 114
4.2.6 Pods horizontal skalieren ... 115
4.2.7 Einen Replikationscontroller löschen ... 117
4.3 Replikationssätze anstelle von Replikationscontrollern verwenden ... 118
4.3.1 Replikationssätze und Replikationscontroller im Vergleich ... 118
4.3.2 Einen Replikationssatz definieren ... 119
4.3.3 Einen Replikationssatz erstellen und untersuchen ... 120

4.3.4 Die ausdrucksstärkeren Labelselektoren des Replikationssatzes .. 121
4.3.5 Zusammenfassung: Replikationssätze 122
4.4 Daemonsets zur Ausführung einer Instanz eines Pods auf jedem Knoten .. 122
4.4.1 Einen Pod auf allen Knoten ausführen 122
4.4.2 Einen Pod nur auf einigen Knoten ausführen 123
4.5 Pods für endliche Aufgaben 126
4.5.1 Jobs 127
4.5.2 Einen Job definieren 128
4.5.3 Ein Job in Aktion 128
4.5.4 Mehrere Podinstanzen in einem Job ausführen 129
4.5.5 Die Zeit zum Abschließen eines Job-Pods begrenzen 130
4.6 Jobs regelmäßig oder zu einem späteren Zeitpunkt ausführen 131
4.6.1 Einen Cron-Job erstellen 131
4.6.2 Die Ausführung geplanter Jobs 132
4.7 Zusammenfassung 133

5 Dienste: Pods finden und mit ihnen kommunizieren 135

5.1 Dienste 136
5.1.1 Dienste erstellen 137
5.1.2 Dienste finden 143
5.2 Verbindungen zu Diensten außerhalb des Clusters 147
5.2.1 Dienstendpunkte 147
5.2.2 Manuell eingerichtete Dienstendpunkte 148
5.2.3 Einen Alias für einen externen Dienst erstellen 150
5.3 Dienste für externe Clients verfügbar machen 151
5.3.1 Einen NodePort-Dienst verwenden 151
5.3.2 Einen Dienst über einen externen Load Balancer verfügbar machen 155
5.3.3 Besondere Eigenschaften von externen Verbindungen 157
5.4 Dienste über eine Ingress-Ressource extern verfügbar machen 159
5.4.1 Eine Ingress-Ressource erstellen 161
5.4.2 Über den Ingress auf den Dienst zugreifen 162
5.4.3 Mehrere Dienste über denselben Domänennamen verfügbar machen 163
5.4.4 Einen Ingress für TLS-Datenverkehr einrichten 164
5.5 Die Bereitschaft eines Pods zur Annahme von Verbindungen signalisieren 166
5.5.1 Bereitschaftssonden 167
5.5.2 Einem Pod eine Bereitschaftssonde hinzufügen 168
5.5.3 Bereitschaftssonden in der Praxis 170
5.6 Headless-Dienste zur Ermittlung einzelner Pods 172
5.6.1 Einen headless-Dienst erstellen 172
5.6.2 Pods über DNS finden 173
5.6.3 Alle Pods finden – auch diejenigen, die nicht bereit sind 174

5.7 Fehlerbehebung bei Diensten ... 175
5.8 Zusammenfassung ... 176

6 Volumes: Festplattenspeicher zu Containern hinzufügen ... 177
6.1 Volumes ... 178
6.1.1 Ein Beispiel ... 178
6.1.2 Arten von Volumes ... 180
6.2 Gemeinsame Datennutzung durch die Container ... 181
6.2.1 emptyDir-Volumes ... 181
6.2.2 Ein Git-Repository als Ausgangspunkt für ein Volume verwenden 184
6.3 Zugriff auf Dateien im Dateisystem des Arbeitsknotens ... 187
6.3.1 HostPath-Volumes ... 188
6.3.2 Systempods mit hostPath-Volumes ... 188
6.4 Dauerhafte Speicherung ... 190
6.4.1 Eine GCE Persistent Disk in einem Pod-Volume ... 190
6.4.2 Andere Arten von Volumes mit zugrunde liegendem persistenten Speicher ... 193
6.5 Pods von der zugrunde liegenden Speichertechnologie entkoppeln ... 195
6.5.1 Persistente Volumes und Claims ... 195
6.5.2 Ein persistentes Volume erstellen ... 196
6.5.3 Mit einem Claim ein persistentes Volume beanspruchen ... 198
6.5.4 Einen Claim in einem Pod verwenden ... 200
6.5.5 Vorteile der Verwendung von persistenten Volumes und Claims .. 201
6.5.6 Persistente Volumes wiederverwenden ... 202
6.6 Persistente Volumes dynamisch bereitstellen ... 204
6.6.1 Die verfügbaren Speichertypen mit Speicherklassen definieren .. 204
6.6.2 Die Speicherklasse in einem Claim angeben ... 205
6.6.3 Dynamische Bereitstellung ohne Angabe einer Speicherklasse ... 207
6.7 Zusammenfassung ... 210

7 Konfigurationszuordnungen und Secrets: Anwendungen konfigurieren ... 213
7.1 Konfiguration von Anwendungen im Allgemeinen ... 213
7.2 Befehlszeilenargumente an Container übergeben ... 215
7.2.1 Den Befehl und die Argumente in Docker definieren ... 215
7.2.2 Den Befehl und die Argumente in Kubernetes überschreiben 217
7.3 Umgebungsvariablen für einen Container einrichten ... 219
7.3.1 Eine Umgebungsvariable in einer Containerdefinition festlegen .. 220
7.3.2 Im Wert einer Variablen auf andere Umgebungsvariablen verweisen ... 220
7.3.3 Die Nachteile hartkodierter Umgebungsvariablen ... 221
7.4 Die Konfiguration mit einer Konfigurationszuordnung entkoppeln ... 221
7.4.1 Einführung in Konfigurationszuordnungen ... 221
7.4.2 Eine Konfigurationszuordnung erstellen ... 223

7.4.3 Einen Konfigurationseintrag als Umgebungsvariable an einen Container übergeben ... 226
7.4.4 Alle Einträge einer Konfigurationszuordnung auf einmal als Umgebungsvariablen übergeben ... 227
7.4.5 Einen Konfigurationseintrag als Befehlszeilenargument übergeben ... 228
7.4.6 Konfigurationsdateien mithilfe eines configMap-Volumes verfügbar machen ... 229
7.4.7 Die Konfiguration einer Anwendung ohne Neustart ändern ... 235
7.5 Sensible Daten mithilfe von Geheimnissen an Container übergeben ... 237
7.5.1 Einführung in Geheimnisse ... 238
7.5.2 Das Geheimnis default-token ... 238
7.5.3 Ein Geheimnis erstellen ... 240
7.5.4 Unterschiede zwischen Konfigurationszuordnungen und Geheimnissen ... 241
7.5.5 Das Geheimnis in einem Pod verwenden ... 243
7.5.6 Geheimnisse zum Abrufen von Images ... 247
7.6 Zusammenfassung ... 248

8 Von Anwendungen aus auf Podmetadaten und andere Ressourcen zugreifen ... 249
8.1 Metadaten über die Downward-API übergeben ... 249
8.1.1 Die verwendbaren Metadaten ... 250
8.1.2 Metadaten über Umgebungsvariablen verfügbar machen ... 251
8.1.3 Metadaten über Dateien in einem downwardAPI-Volume übergeben ... 254
8.2 Kommunikation mit dem Kubernetes-API-Server ... 257
8.2.1 Die REST-API von Kubernetes ... 258
8.2.2 Von einem Pod aus mit dem API-Server kommunizieren ... 262
8.2.3 Botschaftercontainer zur Vereinfachung der Kommunikation mit dem API-Server ... 268
8.2.4 Clientbibliotheken zur Kommunikation mit dem API-Server ... 270
8.3 Zusammenfassung ... 273

9 Deployments: Anwendungen deklarativ aktualisieren ... 275
9.1 Anwendungen in Pods aktualisieren ... 276
9.1.1 Alte Pods löschen und anschließend durch neue ersetzen ... 277
9.1.2 Neue Pods starten und danach die alten löschen ... 277
9.2 Automatische schrittweise Aktualisierung mit einem Replikationscontroller ... 279
9.2.1 Die ursprüngliche Version der Anwendung ausführen ... 279
9.2.2 Die schrittweise Aktualisierung mit kubectl durchführen ... 281
9.2.3 Warum ist kubectl rolling-update veraltet? ... 285
9.3 Deployments zur deklarativen Verwaltung von Anwendungen ... 287

9.3.1 Ein Deployment erstellen ... 287
9.3.2 Ein Deployment aktualisieren ... 290
9.3.3 Eine Bereitstellung zurücknehmen ... 294
9.3.4 Die Rolloutrate festlegen ... 297
9.3.5 Den Rolloutvorgang anhalten ... 299
9.3.6 Das Rollout fehlerhafter Versionen verhindern ... 301
9.4 Zusammenfassung ... 306

10 StatefulSets: Replizierte statusbehaftete Anwendungen bereitstellen ... 307
10.1 Statusbehaftete Pods replizieren ... 307
10.1.1 Mehrere Replikate mit jeweils eigenem Speicher ausführen ... 308
10.1.2 Eine unveränderliche Identität für jeden Pod bereitstellen ... 309
10.2 Statussätze ... 311
10.2.1 Statussätze und Replikationssätze im Vergleich ... 311
10.2.2 Unveränderliche Netzwerkidentität ... 312
10.2.3 Eigenen beständigen Speicher für jede Podinstanz zuweisen ... 314
10.2.4 Garantien von Statussätzen ... 316
10.3 Statussätze nutzen ... 317
10.3.1 Die Anwendung und das Containerimage erstellen ... 317
10.3.2 Die Anwendung mithilfe eines Statussatzes bereitstellen ... 318
10.3.3 Die Pods untersuchen ... 323
10.4 Peers im Statussatz finden ... 327
10.4.1 Die Peer-Ermittlung über DNS einrichten ... 329
10.4.2 Einen Statussatz aktualisieren ... 330
10.4.3 Den Clusterdatenspeicher ausprobieren ... 331
10.5 Umgang mit Knotenausfällen ... 332
10.5.1 Die Trennung eines Knotens vom Netzwerk simulieren ... 332
10.5.2 Den Pod manuell löschen ... 334
10.6 Zusammenfassung ... 336

Teil III: Fortgeschrittene Themen ... 337

11 Interne Mechanismen von Kubernetes ... 339
11.1 Die Architektur ... 339
11.1.1 Die verteilte Natur der Kubernetes-Komponenten ... 340
11.1.2 Verwendung von etcd ... 343
11.1.3 Aufgaben des API-Servers ... 346
11.1.4 Benachrichtigungen des API-Servers über Ressourcenänderungen 348
11.1.5 Der Scheduler ... 350
11.1.6 Die Controller im Controller-Manager ... 352
11.1.7 Die Rolle des Kubelets ... 357
11.1.8 Die Rolle des Kubernetes-Dienstproxys ... 358
11.1.9 Kubernetes-Add-ons ... 359
11.1.10 Zusammenfassung ... 361

11.2 Kooperation der Controller ... 361
11.2.1 Die betroffenen Komponenten ... 361
11.2.2 Die Abfolge der Ereignisse ... 362
11.2.3 Clusterereignisse beobachten ... 364
11.3 Laufende Pods ... 365
11.4 Das Podnetzwerk ... 366
11.4.1 Anforderungen an das Netzwerk ... 366
11.4.2 Funktionsweise des Netzwerks ... 368
11.4.3 CNI ... 370
11.5 Implementierung von Diensten ... 370
11.5.1 Der Kube-Proxy ... 370
11.5.2 Iptables-Regeln ... 371
11.6 Cluster mit hoher Verfügbarkeit ... 373
11.6.1 Anwendungen hochverfügbar machen ... 373
11.6.2 Die Komponenten der Kubernetes-Steuerebene hochverfügbar machen ... 374
11.7 Zusammenfassung ... 377

12 Sicherheit des Kubernetes-API-Servers ... 379
12.1 Authentifizierung ... 379
12.1.1 Benutzer und Gruppen ... 380
12.1.2 Dienstkonten ... 381
12.1.3 Dienstkonten erstellen ... 382
12.1.4 Ein Dienstkonto mit einem Pod verknüpfen ... 384
12.2 Rollengestützte Zugriffssteuerung ... 386
12.2.1 Das RBAC-Autorisierungs-Plug-in ... 386
12.2.2 RBAC-Ressourcen ... 388
12.2.3 Rollen und Rollenbindungen ... 391
12.2.4 Clusterrollen und Clusterrollenbindungen ... 395
12.2.5 Standardclusterrollen und -clusterrollenbindungen ... 404
12.2.6 Berechtigungen bedachtsam gewähren ... 406
12.3 Zusammenfassung ... 407

13 Sicherheit der Clusterknoten und des Netzwerks ... 409
13.1 Die Namespaces des Hostknotens in einem Pod verwenden ... 409
13.1.1 Den Netzwerknamespace des Knotens in einem Pod verwenden ... 410
13.1.2 Bindung an einen Hostport ohne Verwendung des Host-Netzwerknamespace ... 411
13.1.3 Den PID- und den IPC-Namespace des Knotens verwenden ... 413
13.2 Den Sicherheitskontext eines Containers einrichten ... 414
13.2.1 Einen Container unter einer bestimmten Benutzer-ID ausführen ... 415
13.2.2 Die Ausführung eines Containers als root verhindern ... 416
13.2.3 Pods im privilegierten Modus ausführen ... 417
13.2.4 Einem Container einzelne Kernelfähigkeiten hinzufügen ... 418

13.2.5 Fähigkeiten von einem Container entfernen ... 420
13.2.6 Prozesse am Schreiben im Dateisystem des Containers hindern .. 421
13.2.7 Gemeinsame Nutzung von Volumes durch Container mit verschiedenen Benutzer-IDs ... 422
13.3 Die Bearbeitung der Sicherheitsmerkmale in einem Pod einschränken 424
13.3.1 Podsicherheitsrichtlinien ... 424
13.3.2 Die Richtlinien runAsUser, fsGroup und supplementalGroups 427
13.3.3 Zulässige, unzulässige und Standardfähigkeiten festlegen ... 429
13.3.4 Die verwendbaren Arten von Volumes einschränken ... 430
13.3.5 Benutzern und Gruppen unterschiedliche Podsicherheitsrichtlinien zuweisen ... 431
13.4 Das Podnetzwerk isolieren ... 434
13.4.1 Die Netzwerkisolierung in einem Namespace aktivieren ... 435
13.4.2 Einzelnen Pods im Namespace die Verbindung zu einem Serverpod erlauben ... 435
13.4.3 Das Netzwerk zwischen Kubernetes-Namespaces isolieren ... 436
13.4.4 Verwendung der CIDR-Schreibweise zur Isolierung ... 438
13.4.5 Den ausgehenden Datenverkehr von Pods einschränken ... 438
13.5 Zusammenfassung ... 439

14 Die Computerressourcen eines Pods verwalten ... 441
14.1 Ressourcen für die Container eines Pods anfordern ... 441
14.1.1 Pods mit Ressourcenanforderungen erstellen ... 442
14.1.2 Einfluss der Ressourcenanforderungen auf die Zuteilung zu Knoten ... 443
14.1.3 Der Einfluss der CPU-Anforderungen auf die CPU-Zeitzuteilung .. 448
14.1.4 Benutzerdefinierte Ressourcen definieren und anfordern ... 448
14.2 Die verfügbaren Ressourcen für einen Container einschränken ... 449
14.2.1 Harte Grenzwerte für die von einem Container verwendeten Ressourcen festlegen ... 449
14.2.2 Überschreiten der Grenzwerte ... 451
14.2.3 Grenzwerte aus der Sicht der Anwendungen in den Containern .. 452
14.3 QoS-Klassen für Pods ... 454
14.3.1 Die QoS-Klasse eines Pods festlegen ... 454
14.3.2 Auswahl des zu beendenden Prozesses bei zu wenig Speicher 457
14.4 Standardanforderungen und -grenzwerte für die Pods in einem Namespace festlegen ... 458
14.4.1 Der Grenzwertbereich ... 459
14.4.2 Einen Grenzwertbereich erstellen ... 460
14.4.3 Die Grenzwerte durchsetzen ... 461
14.4.4 Standardanforderungen und -grenzwerte anwenden ... 462
14.5 Die in einem Namespace insgesamt verfügbaren Ressourcen beschränken 463
14.5.1 Ressourcenkontingente ... 463
14.5.2 Kontingente für persistenten Speicher festlegen ... 465

14.5.3 Die Höchstzahl der Objekte in einem Namespace beschränken ... 466
14.5.4 Kontingente für einzelne Podstatus und QoS-Klassen festlegen ... 467
14.6 Die Ressourcennutzung der Pods überwachen ... 468
14.6.1 Die tatsächliche Ressourcennutzung erfassen ... 468
14.6.2 Verlaufsdaten des Ressourcenverbrauchs speichern und analysieren ... 470
14.7 Zusammenfassung ... 474

15 Automatische Skalierung von Pods und Clusterknoten ... 475
15.1 Automatische horizontale Podskalierung ... 476
15.1.1 Der Vorgang der automatischen Skalierung ... 476
15.1.2 Skalierung auf der Grundlage der CPU-Nutzung ... 479
15.1.3 Skalierung auf der Grundlage der Speichernutzung ... 486
15.1.4 Skalierung auf der Grundlage anderer Messgrößen ... 486
15.1.5 Geeignete Messgrößen für die automatische Skalierung auswählen ... 488
15.1.6 Herunterskalieren auf null Replikate ... 489
15.2 Automatische vertikale Podskalierung ... 489
15.2.1 Ressourcenanforderungen automatisch einrichten ... 489
15.2.2 Ressourcenanforderungen von laufenden Pods ändern ... 490
15.3 Horizontale Skalierung von Clusterknoten ... 490
15.3.1 Der Cluster-Autoskalierer ... 490
15.3.2 Den Cluster-Autoskalierer aktivieren ... 492
15.3.3 Die Unterbrechung von Diensten beim Herunterskalieren des Clusters minimieren ... 493
15.4 Zusammenfassung ... 495

16 Erweiterte Planung ... 497
16.1 Pods mithilfe von Mängeln und Tolerierungen von bestimmten Knoten fernhalten ... 497
16.1.1 Mängel und Tolerierungen ... 498
16.1.2 Einem Knoten benutzerdefinierte Mängel hinzufügen ... 500
16.1.3 Tolerierungen zu Pods hinzufügen ... 500
16.1.4 Verwendungszwecke für Mängel und Tolerierungen ... 501
16.2 Knotenaffinität ... 502
16.2.1 Feste Knotenaffinitätsregeln aufstellen ... 503
16.2.2 Knotenprioritäten bei der Zuteilung eines Pods ... 505
16.3 Pods mit Affinitäts- und Antiaffinitätsregeln auf denselben Knoten unterbringen ... 508
16.3.1 Podaffinitätsregeln zur Bereitstellung von Pods auf demselben Knoten ... 509
16.3.2 Pods im selben Schaltschrank, in derselben Verfügbarkeitszone oder derselben geografischen Region bereitstellen ... 511
16.3.3 Präferenzen statt fester Regeln für die Podaffinität angeben ... 513

16.3.4 Pods mit Antiaffinitätsregeln voneinander getrennt halten 514
16.4 Zusammenfassung 516

17 Best Practices für die Anwendungsentwicklung 519
17.1 Das Gesamtbild 519
17.2 Der Lebenszyklus eines Pods 521
17.2.1 Beendigung und Verlegung von Anwendungen 521
17.2.2 Tote oder teilweise tote Pods neu bereitstellen 524
17.2.3 Pods in einer bestimmten Reihenfolge starten 525
17.2.4 Lebenszyklushooks 527
17.2.5 Pods herunterfahren 531
17.3 Die ordnungsgemäße Verarbeitung aller Clientanforderungen sicherstellen 535
17.3.1 Unterbrechungen von Clientverbindungen beim Hochfahren eines Pods verhindern 535
17.3.2 Unterbrechungen von Clientverbindungen beim Herunterfahren eines Pods verhindern 535
17.4 Einfache Ausführung und Handhabung von Anwendungen in Kubernetes . 540
17.4.1 Einfach zu handhabende Containerimages erstellen 540
17.4.2 Images sauber kennzeichnen 541
17.4.3 Mehrdimensionale statt eindimensionaler Labels 541
17.4.4 Ressourcen mit Anmerkungen beschreiben 542
17.4.5 Gründe für die Beendigung eines Prozesses angeben 542
17.4.6 Anwendungsprotokolle 544
17.5 Empfohlene Vorgehensweisen für Entwicklung und Tests 546
17.5.1 Anwendungen während der Entwicklung außerhalb von Kubernetes ausführen 546
17.5.2 Minikube für die Entwicklung 547
17.5.3 Versionssteuerung und Manifeste zur automatischen Bereitstellung von Ressourcen 549
17.5.4 Ksonnet als Alternative zu YAML- und JSON-Manifesten 549
17.5.5 Continuous Integration und Continuous Delivery (CI/CD) 550
17.6 Zusammenfassung 551

18 Kubernetes erweitern 553
18.1 Eigene API-Objekte definieren 553
18.1.1 Eigene Ressourcendefinitionen 554
18.1.2 Benutzerdefinierte Ressourcen mit benutzerdefinierten Controllern automatisieren 558
18.1.3 Benutzerdefinierte Objekte validieren 562
18.1.4 Einen benutzerdefinierten API-Server für benutzerdefinierte Objekte bereitstellen 562
18.2 Kubernetes mit dem Kubernetes-Dienstkatalog erweitern 564
18.2.1 Der Dienstkatalog 565

18.2.2 Der API-Server des Dienstkatalogs und der Controller-Manager .. 566
18.2.3 Dienstbroker und die API OpenServiceBroker 567
18.2.4 Dienste bereitstellen und nutzen 568
18.2.5 Aufheben der Bindung und der Bereitstellung 571
18.2.6 Vorteile des Dienstkatalogs 571
18.3 Plattformen auf der Grundlage von Kubernetes 572
18.3.1 Die Containerplattform Red Hat OpenShift 572
18.3.2 Deis Workflow und Helm 575
18.4 Zusammenfassung 578

Anhang A: Verwendung von kubectl für mehrere Cluster 579
A.1 Umschalten zwischen Minikube und Google Kubernetes Engine 579
A.1.1 Umschalten zu Minikube 579
A.1.2 Umschalten zu GKE 579
A.2 Verwendung von kubectl für mehrere Cluster oder Namespaces 580
A.2.1 Den Speicherort der Konfigurationsdatei festlegen 580
A.2.2 Der Inhalt der Konfigurationsdatei 580
A.2.3 Konfigurationseinträge auflisten, hinzufügen und ändern 581
A.2.4 Verwendung von kubectl mit verschiedenen Clustern, Benutzern und Kontexten 583
A.2.5 Umschalten zwischen Kontexten 583
A.2.6 Kontexte und Cluster auflisten 584
A.2.7 Kontexte und Cluster löschen 584

Anhang B: Einen Cluster mit mehreren Knoten mit kubeadm erstellen 585
B.1 Das Betriebssystem und die erforderlichen Pakete einrichten 585
B.1.1 Die virtuelle Maschine erstellen 585
B.1.2 Den Netzwerkadapter für die VM einrichten 586
B.1.3 Das Betriebssystem installieren 587
B.1.4 Docker und Kubernetes installieren 590
B.1.5 Die VM klonen 591
B.2 Den Master mit kubeadm konfigurieren 593
B.2.1 Ausführung der Komponenten durch kubeadm 594
B.3 Arbeitsknoten mit kubeadm einrichten 595
B.3.1 Das Containernetzwerk einrichten 596
B.4 Vom lokalen Computer auf den Cluster zugreifen 597

Anhang C: Andere Containerlaufzeitumgebungen verwenden 599
C.1 Docker durch rkt ersetzen 599
C.1.1 Kubernetes zur Verwendung von rkt einrichten 599
C.1.2 rkt in Minikube ausprobieren 600
C.2 Andere Containerlaufzeiten über die CRI verwenden 602
C.2.1 CRI-O 602
C.2.2 Anwendungen in VMs statt in Containern ausführen 602

Anhang D: Clusterverbund **603**
D.1 Der Kubernetes-Clusterverbund 603
D.2 Die Architektur 604
D.3 Verbund-API-Objekte 605
D.3.1 Verbundversionen der Kubernetes-Ressourcen 605
D.3.2 Funktionsweise von Verbundressourcen 606

Anhang E: Kubernetes-Ressourcen in diesem Buch **609**

Index **613**

Vorwort

Nachdem ich schon einige Jahre für Red Hat gearbeitet hatte, wurde ich Ende 2014 einem neuen Team namens *Cloud Enablement* zugeteilt. Unsere Aufgabe bestand darin, die Middleware-Produktpalette unseres Unternehmens auf die Containerplattform OpenShift zu übertragen, die zu diesem Zeitpunkt auf der Grundlage von Kubernetes entwickelt wurde. Damals steckte Kubernetes noch in den Kinderschuhen – noch nicht einmal Version 1.0 war veröffentlicht worden!

In unserem Team mussten wir uns mit den Interna von Kubernetes schnell vertraut machen, um unsere Software in die richtige Richtung lenken und alle Möglichkeiten ausnutzen zu können, die Kubernetes bot. Wenn wir auf ein Problem stießen, konnten wir oft schlecht unterscheiden, ob wir irgendetwas falsch machten oder ob es an einem der Bugs lag, unter denen Kubernetes in seiner Frühzeit litt.

Sowohl Kubernetes als auch meine Kenntnisse darüber haben sich seitdem erheblich weiterentwickelt. Als ich begann, damit zu arbeiten, hatten die meisten noch nie davon gehört. Heute kennt es praktisch jeder Softwareentwickler, und unter allen Möglichkeiten, um Anwendungen sowohl in der Cloud als auch in Rechenzentren am eigenen Standort auszuführen, gehört es zu denen mit der weitesten Verbreitung und dem schnellsten Wachstum.

Während der ersten Monate meiner Arbeit mit Kubernetes schrieb ich einen zweiteiligen Blogpost darüber, wie man einen JBoss-WildFly-Anwendungsservercluster in OpenShift/Kubernetes ausführt. Damals ahnte ich es noch nicht, aber dieser einfache Post führte schließlich dazu, dass der Verlag Manning mit der Bitte an mich herantrat, ein Buch über Kubernetes zu schreiben. Natürlich konnte ich ein solches Angebot nicht ablehnen, obwohl ich sicher war, dass Manning auch andere mögliche Autoren angesprochen hatte und sich letzten Endes für jemand anderen entscheiden würde.

Wie Sie sehen, geschah das nicht. Nach mehr als anderthalb Jahren Schreib- und Recherchearbeit ist das Buch nun fertig geworden. Es war ein großartiges Erlebnis. Ein Buch über ein technisches Thema zu schreiben ist die beste Möglichkeit, um die betreffende Technologie viel ausführlicher kennenzulernen, als es durch reine Anwendung möglich wäre. Da sich nicht nur meine Kenntnisse über Kubernetes erweitert haben, sondern Kubernetes selbst weiterentwickelt wurde, musste ich ständig bereits fertiggestellte Kapitel umschreiben und ergänzen. Als Perfektionist werde ich niemals absolut zufrieden mit diesem Buch sein, aber ich freue mich zu hören, dass viele Personen, die Vorabversionen über das Manning Early Access Program gelesen haben, es für einen großartigen Leitfaden zum Thema Kubernetes halten.

Mein Ziel bestand darin, den Lesern die Technologie von Kubernetes nahezubringen und den Gebrauch der Werkzeuge für eine wirkungsvolle und rationelle Entwicklung und Bereitstellung von Anwendungen in Kubernetes-Clustern vorzuführen. Die Einrichtung und Wartung eines hochverfügbaren Kubernetes-Clusters ist jedoch kein Schwerpunkt dieses Buchs, allerdings dürfte Ihnen der letzte Teil solide Kenntnisse darüber vermitteln, woraus ein solcher Cluster besteht, sodass Sie andere Quellen zu diesem Thema besser verstehen können.

Ich hoffe, dass Sie die Lektüre genießen können und das Buch Ihnen zeigt, wie Sie den größten Nutzen aus Kubernetes ziehen können.

Der Autor

Marko Lukša ist Softwareentwickler mit mehr als 20 Jahren Berufserfahrung, wobei die Palette seiner Projekte von einfachen Webanwendungen bis zu vollständigen ERP-Systemen, Frameworks und Middleware reicht. Seine ersten Programmierversuche hat er bereits 1985 im Alter von sechs Jahren auf einem ZX Spectrum gemacht, den sein Vater gebraucht für ihn gekauft hatte. In der Grundschule wurde er Landesmeister im Logo-Programmierwettbewerb und nahm an Programmierferienlagern teil, in denen er Pascal lernte. Seitdem hat er Software in einer breiten Palette von Programmiersprachen entwickelt.

In der weiterführenden Schule begann er damit, dynamische Websites zu erstellen, als das Web noch ziemlich jung war. Während seines Studiums der Informatik an der Universität von Ljubljana in Slowenien entwickelte er bei einem ortsansässigen Unternehmen Software für das Gesundheitswesen und die Telekommunikationsbranche. Schließlich begann er für Red Hat zu arbeiten. Dort entwickelte er zu Anfang eine Open-Source-Implementierung der Google App Engine API, die die Middleware JBoss von Red Hat verwendete. Außerdem arbeitete er an Projekten wie CDI/Weld, Infinispan/JBoss Data Grid u. a. mit.

Seit Ende 2014 gehört er zum Cloud-Enablement-Team von Red Hat. Zu seinen Aufgaben gehört es dabei, sich über die neuesten Entwicklungen bei Kubernetes und verwandten Technologien auf dem neuesten Stand zu halten und dafür zu sorgen, dass die Middleware des Unternehmens die Möglichkeiten von Kubernetes und OpenShift voll ausnutzt.

Danksagung zur englischsprachigen Ausgabe

Bevor ich mit dem Schreiben dieses Buches begann, hatte ich keine Vorstellung davon, wie viele Personen daran beteiligt sind, um aus einem ersten Manuskript eine fertige Veröffentlichung zu machen. Es gibt viele Menschen, denen ich Dank schulde.

Als Erstes möchte ich Erin Twohey danken, die mich gebeten hat, dieses Buch zu schreiben, und Michael Stephens, der von Anfang an volles Vertrauen darin gesetzt hat, dass ich es schaffen kann. Seine ermutigenden Worte haben mich zu Anfang stark motiviert und diese Motivation während der letzten anderthalb Jahre aufrechterhalten.

Ich möchte auch meinem ursprünglichen Entwicklungsredakteur Andrew Warren danken, der mir half, das erste Kapitel fertigzustellen, und seiner Nachfolgerin Elesha Hyde, die mit mir danach bis zum letzten Kapitel gearbeitet hat. Vielen Dank dafür, dass sie es mit mir ausgehalten haben, auch wenn der Umgang mit mir nicht ganz leicht ist und ich ziemlich oft einfach vom Radar verschwinde.

Ich möchte auch Jeanne Boyarsky danken, die als erste Lektorin meine Kapitel las und kommentierte, während ich noch daran schrieb. Jeanne und Elesha trugen erheblich dazu bei, das Buch so gut zu machen, wie es hoffentlich ist. Ohne ihre Kommentare hätte das Buch niemals so positive Bewertungen von externen Gutachtern und Lesern bekommen können.

Ich möchte auch meinem Fachlektor Antonio Magnaghi und natürlich allen externen Gutachtern danken: Al Krinker, Alessandro Campeis, Alexander Myltsev, Csaba Sari, David DiMaria, Elias Rangel, Erisk Zelenka, Fabrizio Cucci, Jared Duncan, Keith Donaldson, Michael Bright, Paolo Antinori, Peter Perlepes und Tiklu Ganguly. Ihre positive Rückmeldung hat mich durchhalten lassen, wenn ich manchmal das Gefühl hatte, mein Text sei fürchterlich geschrieben und völlig nutzlos, und ihre konstruktive Kritik hat mir geholfen, die Abschnitte zu verbessern, die ich ohne große Anstrengung flott zusammengestoppelt hatte. Vielen Dank dafür, dass Sie mich auf schwer verständliche Stellen hingewiesen und Vorschläge zur Verbesserung des Buches gemacht haben. Vielen Dank auch dafür, die richtigen Fragen zu stellen, die mir deutlich machten, dass ich ein oder zwei Dinge in der ursprünglichen Version meines Manuskripts falsch dargestellt hatte.

Ich möchte auch den Lesern danken, die Vorabversionen dieses Buches über das Early-Access-Programm von Manning (MEAP) erworben und ihre Kommentare im Onlineforum abgegeben oder mich direkt angesprochen haben, insbesondere Vimal Kansal, Paolo Patierno und Roland Huß, die einige Inkonsistenzen und andere Fehler gefunden haben. Des Weiteren möchte ich allen Manning-Mitarbeitern danken, die an der Produktion dieses Buches beteiligt waren. Bevor ich zum Schluss komme, möchte ich meinem Kollegen und Schulfreund Aleš Justin danken, der mich zu Red Hat gebracht hat, und meinen wunderbaren Kollegen im Cloud-Enablement-Team. Wäre ich nicht bei Red Hat und in diesem Team gewesen, so wäre ich nicht derjenige gewesen, der dieses Buch geschrieben hat.

Abschließend möchte ich meiner Frau und meinem Sohn danken, die während der letzten 18 Monate mehr als verständnisvoll waren und mich unterstützt haben, obwohl ich mich in mein Büro verkrochen habe, anstatt Zeit mit ihnen zu verbringen.

Vielen Dank euch allen!

Über dieses Buch

Dieses Buch soll Sie zu einem kompetenten Kubernetes-Benutzer machen. Sie lernen hier praktisch alle Prinzipien kennen, die Sie beherrschen müssen, um Anwendungen in einer Kubernetes-Umgebung zu entwickeln und auszuführen.

Bevor es mit Kubernetes losgeht, erhalten Sie einen Überblick über Containertechnologien wie Docker und das Erstellen von Containern, damit Sie den weiteren Ausführungen auch dann folgen können, wenn Sie damit noch nicht gearbeitet haben. Danach werden Sie Schritt für Schritt in alles eingeführt, was Sie über Kubernetes wissen müssen, von den Grundprinzipien zu den verborgenen Mechanismen.

Zielgruppe

Dieses Buch richtet sich hauptsächlich an Anwendungsentwickler, bietet aber auch einen Überblick über die Verwaltung von Anwendungen. Es ist für alle gedacht, die sich für die Ausführung und Verwaltung von Containeranwendungen auf mehr als einem einzigen Server interessieren.

Sowohl Anfänger als auch erfahrene Softwareentwickler, die etwas über Containertechnologien lernen möchten, erhalten hier die notwendigen Kenntnisse, um ihre Anwendungen in einer Kubernetes-Umgebung zu entwickeln und in Containern auszuführen.

Vorkenntnisse in Containertechnologien und Kubernetes sind nicht erforderlich. Die Erklärungen in diesem Buch bauen aufeinander auf, und es wird kein Beispielcode verwendet, der nur für Experten verständlich wäre.

Leser sollten allerdings Grundkenntnisse in Programmierung, Computernetzwerken, einfachen Linux-Befehlen und Standardprotokollen wie HTTP mitbringen.

Der Aufbau dieses Buches

Dieses Buch besteht aus 18 Kapiteln, die in drei Teile gegliedert sind.

Teil 1 gibt eine kurze Einführung in Docker und Kubernetes. Sie erfahren hier, wie Sie einen Kubernetes-Cluster einrichten und darin eine einfache Anwendung ausführen. Dieser Teil umfasst nur zwei Kapitel:

- Kapitel 1 erklärt, was Kubernetes ist, wie es entstand und wie es hilft, die heutigen Probleme der Verwaltung und Skalierung von Anwendungen zu lösen.
- Kapitel 2 enthält eine praktische Anleitung, um ein Containerimage zu erstellen und in einem Kubernetes-Cluster auszuführen. Es erklärt auch, wie Sie lokale Kubernetes-Cluster mit einem Knoten und richtige Cluster mit mehreren Knoten in der Cloud ausführen.

Teil 2 stellt die Grundprinzipien vor, mit denen Sie vertraut sein müssen, um Anwendungen in Kubernetes auszuführen. Er besteht aus folgenden Kapiteln:

- Kapitel 3 stellt die Grundbausteine von Kubernetes vor – die Pods – und erklärt, wie Pods und andere Kubernetes-Objekte mithilfe von Labels geordnet werden können.
- Kapitel 4 erklärt, wie Kubernetes Anwendungen durch den automatischen Neustart von Containern in funktionsfähigem Zustand hält. Hier erfahren Sie auch, wie Sie verwaltete Pods ausführen, horizontal skalieren, gegen den Ausfall von Clusterknoten absichern und zu vorherbestimmten Zeitpunkten oder regelmäßig ausführen.
- Kapitel 5 zeigt, wie Pods ihre Dienste für Clients im und außerhalb des Clusters verfügbar machen und wie Pods im Cluster Dienste innerhalb oder außerhalb des Clusters entdecken und nutzen können.
- Kapitel 6 erklärt, wie mehrere Container im selben Pod Dateien gemeinsam nutzen können und wie Sie persistenten Speicher verwalten und für die Pods zugänglich machen.
- Kapitel 7 zeigt, wie Sie Konfigurationsdaten und sensible Informationen, z. B. Anmeldeinformationen, an Anwendungen innerhalb der Pods übergeben.
- Kapitel 8 beschreibt, wie Anwendungen Informationen über die Kubernetes-Umgebung beziehen, in der sie laufen, und wie sie mit Kubernetes kommunizieren, um den Status des Clusters zu ändern.
- Kapitel 9 gibt eine Einführung in das Prinzip von Deployments und erklärt, wie Sie Anwendungen in einer Kubernetes-Umgebung ordnungsgemäß ausführen und aktualisieren.
- Kapitel 10 stellt eine Möglichkeit zur Ausführung statusbehafteter Anwendungen vor, die gewöhnlich eine stabile Identität benötigen und ihren Status erhalten müssen.

In Teil 3 geht es um die internen Mechanismen von Kubernetes-Clustern. Es werden hier nicht nur einige neue Konzepte eingeführt, sondern auch alle Funktionsprinzipien, die Sie in den ersten beiden Teilen gelernt haben, von einer höheren Warte aus untersucht. Dieser Teil umfasst folgende Kapitel:

- Kapitel 11 stößt unter die Oberfläche von Kubernetes vor und beschreibt die Komponenten, aus denen ein Kubernetes-Cluster besteht, und ihre Funktionsweise. Außerdem wird hier erklärt, wie Pods über das Netzwerk kommunizieren und wie Dienste die Last auf mehrere Pods verteilen.

- Kapitel 12 erklärt, wie Sie Ihren Kubernetes-API-Server und den Cluster mithilfe von Authentifizierung und Autorisierung sicherer gestalten können.
- Kapitel 13 zeigt, wie Pods auf die Ressourcen des Knotens zugreifen und wie ein Clusteradministrator sie daran hindern kann.
- Kapitel 14 beschreibt, wie Sie den Verbrauch von Computerressourcen durch die einzelne Anwendung einschränken, die garantierte Dienstqualität der Anwendungen festlegen, die Ressourcennutzung überwachen und Benutzer daran hindern, zu viele Ressourcen zu verbrauchen.
- Kapitel 15 erklärt, wie Sie Kubernetes einrichten, um die Anzahl der Replikate einer Anwendung automatisch skalieren zu lassen, und wie Sie die Größe des Clusters erhöhen, wenn die vorhandenen Clusterknoten keine weiteren Anwendungen mehr aufnehmen können.
- Kapitel 16 zeigt, wie Sie dafür sorgen, dass Pods bestimmten Knoten zugeteilt oder nicht zugeteilt werden. Außerdem erfahren Sie, wie Sie Pods zusammen oder getrennt zuteilen können.
- Kapitel 17 beschreibt, wie Sie Ihre Anwendungen auf clustergerechte Weise entwickeln sollten. Sie erhalten dabei auch einige Hinweise dazu, wie Sie bei der Entwicklung und dem Testen vorgehen sollten, um Störungen zu vermeiden.
- Kapitel 18 zeigt, wie Sie Kubernetes mit eigenen Objekten erweitern können und wie andere dies bereits getan und damit professionelle Anwendungsplattformen erstellt haben.

Bei der Lektüre werden Sie nicht nur die einzelnen Bestandteile von Kubernetes kennenlernen, sondern auch Ihre Kenntnisse des Befehlszeilentools `kubectl` nach und nach erweitern.

Der Code

Dieses Buch enthält nicht viel Quellcode, aber dafür eine Menge Manifeste für Kubernetes-Ressourcen im YAML-Format sowie Shellbefehle und deren Ausgabe. All diese Elemente sind in `nichtproportionaler Schrift` dargestellt, um sie vom normalen Fließtext abzuheben.

Die Shellbefehle sind gewöhnlich in fetter nichtproportionaler Schrift angegeben, um sie von der Ausgabe zu unterscheiden. Manchmal sind jedoch nur die wichtigsten Teile eines Befehls oder auch einige besondere Teile der Ausgabe fett hervorgehoben. Die Ausgabe wurde meistens umformatiert, sodass sie in eine Buchzeile passt. Da das Kubernetes-Befehlszeilenwerkzeug `kubectl` ständig weiterentwickelt wird, kann es sein, dass sich die Ausgabe neuerer Versionen von dem unterscheidet, was Sie in diesem Buch sehen. Die Listings enthalten oft auch Anmerkungen, die die wichtigsten Teile hervorheben und erklären.

Alle Beispiele in diesem Buch wurden mit Kubernetes 1.8 in der Google Kubernetes Engine und in einem lokalen Minikube-Cluster getestet. Der vollständige Quellcode und die YAML-Manifeste sind auf *https://github.com/luksa/kubernetes-in-action* zu finden.

Das Forum zum Buch

Manning Publications, der Herausgeber der Originalausgabe, unterhält ein Forum (in englischer Sprache), in dem Sie das Buch kommentieren, fachliche Fragen stellen und Hilfe sowohl vom Autor als auch von anderen Lesern erhalten können. Dieses Forum finden Sie auf *https://forums.manning.com/forums/kubernetes-in-action*. Um mehr über die Manning-Foren und die Verhaltensregeln dafür zu erfahren, schauen Sie auf *https://forums.manning.com/forums/about* nach.

Manning möchte damit eine Möglichkeit für den Austausch zwischen Lesern und zwischen Leser und Autor geben. Der Autor ist dabei in keiner Form zu irgendeiner garantierten Form der Beteiligung verpflichtet, da er alle seine Beiträge freiwillig (und unbezahlt) leistet. Wir raten Ihnen, anspruchsvolle Fragen zu stellen, falls sein Interesse nachlassen sollte. Das Forum und die Archive früherer Diskussionen sind auf der Website von Manning so lange zugänglich, wie die Originalausgabe dieses Buches in Druck ist.

Sonstige Onlinequellen

Informationen über Kubernetes sind auch an folgenden Orten zu finden:

- Auf der Kubernetes-Website *https://kubernetes.io*
- Im Kubernetes-Blog auf *http://blog.kubernetes.io*, in dem regelmäßig interessante Informationen erscheinen
- Im Slack-Kanal der Kubernetes-Community auf *http://slack.k8s.io*
- In den YouTube-Kanälen von Kubernetes und der Cloud Native Computing Foundation:
- *https://www.youtube.com/channel/UCZ2bu0qutTOM0tHYa_jkIwg*
- *https://www.youtube.com/channel/UCvqbFHwN-nwalWPjPUKpvTA*

Um mehr über einzelne Themen zu erfahren oder auch um selbst zu Kubernetes beitragen zu können, wenden Sie sich an die Kubernetes-Interessengruppen (Special Interest Groups, SIGs) auf *https://github.com/kubernetes/kubernetes/wiki/Special-Interest-Groups-(SIGs)*.

Da es sich bei Kubernetes um Open-Source-Software handelt, enthält auch der Kubernetes-Quellcode einen Schatz an Informationen. Sie finden ihn in *https://github.com/kubernetes/kubernetes* und verwandten Repositorys.

3 Pods: Container in Kubernetes ausführen

Der Inhalt dieses Kapitels

- Pods erstellen, ausführen und anhalten
- Pods und andere Ressourcen mit Labels ordnen
- Eine Operation an allen Pods mit einem gegebenen Label durchführen
- Pods mithilfe von Namespaces in nicht überlappende Gruppen aufteilen
- Container bestimmten Arten von Arbeitsknoten zuweisen

Im vorherigen Kapitel haben Sie einen groben Überblick über die grundlegenden Komponenten erhalten, die Sie in Kubernetes erstellen, und zumindest in Umrissen gesehen, was sie tun. Jetzt wollen wir alle Arten von Kubernetes-Objekten (oder *Ressourcen*) ausführlich darstellen, damit Sie lernen, wann, wie und warum Sie sie jeweils einsetzen müssen. Dabei beginnen wir mit den *Pods*, denn dies sind die zentralen, wichtigsten Objekte in Kubernetes. Alle anderen dienen dazu, die Pods zu verwalten oder zu exponieren, oder werden von ihnen verwendet.

3.1 Einführung in Pods

Wie wir bereits gesehen haben, ist ein Pod eine Gruppe von Containern, die sich auf demselben Knoten befinden, und stellt den Grundbaustein von Kubernetes dar. Wir stellen nicht einzelne Container bereit, sondern immer Pods. Das heißt nicht, dass Pods grundsätzlich mehr als einen Container enthalten. Sehr häufig schließt ein Pod nur einen einzigen Container ein. Wenn ein Pod aber tatsächlich über mehrere Container verfügt, dann werden diese immer auf einem einzigen Arbeitsknoten ausgeführt. Ein Pod überspannt niemals mehrere Knoten, wie Bild 3.1 zeigt.

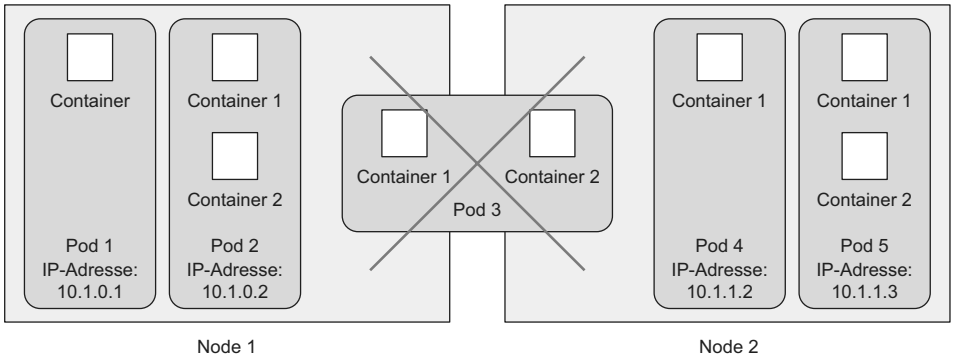


Bild 3.1 Alle Container eines Pods werden auf demselben Knoten ausgeführt. Ein Pod überspannt niemals mehrere Knoten.

3.1.1 Wozu benötigen wir Pods?

Aber wozu brauchen wir überhaupt Pods? Warum können wir die Container nicht einfach direkt verwenden? Warum müssen wir überhaupt mehrere Container nebeneinander verwenden? Können wir nicht einfach alle Prozesse in einen einzigen Container packen? Diese Fragen wollen wir im Folgenden beantworten.

Warum mehrere Container besser sind als ein Container mit mehreren Prozessen

Stellen Sie sich eine Anwendung aus mehreren Prozessen vor, die über IPC *(Inter-Process Communication)* oder lokal gespeicherte Dateien miteinander kommunizieren. Das macht es erforderlich, dass sie auf demselben Computer laufen. Da Prozesse in Kubernetes immer in Containern ausgeführt werden und jeder Container sehr wahrscheinlich auf einem eigenen Rechner läuft, scheint es sinnvoll zu sein, mehrere Prozesse in einem einzigen Container auszuführen. Trotzdem sollten Sie das nicht tun.

Container sind zur Ausführung eines einzigen Prozesses ausgelegt (abgesehen davon, dass der Prozess eigene Kindprozesse erzeugen kann). Wenn Sie mehrere nicht zusammengehörige Prozesse in einem einzigen Container ausführen, müssen Sie selbst dafür sorgen, dass all diese Prozesse laufen, ihre Protokolle verwalten usw. Beispielsweise müssten Sie einen Mechanismus hinzufügen, damit einzelne Prozesse automatisch neu gestartet werden, wenn sie abstürzen. Des Weiteren würden alle Prozesse ihre Protokolle in denselben Standardausgang schreiben, sodass es ziemlich schwer wäre herauszufinden, welcher Prozess was protokolliert hat.

Daher müssen Sie jeden Prozess in seinem eigenen Container ausführen. Das ist die vorgesehene Methode zur Nutzung von Docker und Kubernetes.

3.1.2 Grundlagen von Pods

Da es nicht vorgesehen ist, in einem Container mehrere Prozesse unterzubringen, brauchen wir offensichtlich einen Mechanismus, um mehrere Container aneinander zu binden und als eine Einheit zu behandeln. Das ist die Begründung für die Verwendung von Pods.

Ein Containerpod ermöglicht es, mehrere eng miteinander verbundene Prozesse gemeinsam auszuführen und mit (fast) derselben Umgebung zu versehen, als ob sie alle in einem einzigen Container ausgeführt werden, und sie gleichzeitig bis zu einem gewissen Grad voneinander zu isolieren. Dadurch können wir sowohl die Vorteile von Containern nutzen als auch den Prozessen vorgaukeln, dass sie zusammen ausgeführt werden.

Teilisolierung zwischen den Containern in einem Pod

Im vorherigen Kapitel haben wir gesagt, dass Container vollständig voneinander isoliert sind. In Wirklichkeit jedoch wollen wir nicht einzelne Container, sondern Gruppen von Containern voneinander isolieren. Die Container innerhalb einer Gruppe sollen sich einige Ressourcen teilen können, allerdings nicht alle. Das bedeutet, dass sie nicht vollständig isoliert sind. Kubernetes erreicht dies durch eine Konfiguration von Docker, sodass sich alle Container in einem Pod dieselben Linux-Namespaces teilen, anstatt jeweils ihre eigenen zu verwenden.

Da alle Container in einem Pod in denselben *Netzwerk*- und *UTS*-Namespaces ausgeführt werden (wir sprechen hier über Linux-Namespaces), teilen sie sich auch alle denselben Hostnamen und dieselben Netzwerkschnittstellen. Außerdem befinden sich alle Container eines Pods im selben *IPC*-Namespace und können daher über IPC miteinander kommunizieren. Sie sollten auch denselben *PID*-Namespace verwenden, allerdings ist dieses Merkmal nicht standardmäßig aktiviert.

HINWEIS: Da die Container eines Pods zurzeit verschiedene PID-Namespaces verwenden, können Sie die eigenen Prozesse eines Containers nur einsehen, wenn Sie `ps aux` im Container ausführen.

Beim Dateisystem sieht die Sache jedoch anders aus. Da der Großteil des Container-Dateisystems von dem Containerimage stammt, ist das Dateisystem jedes Containers standardmäßig vollständig von den Dateisystemen anderer Container getrennt. Mithilfe von *Volumes*, einer weiteren Art von Kubernetes-Ressource, können sie jedoch Dateiverzeichnisse gemeinsam nutzen. Darüber werden wir in Kapitel 6 sprechen.

Gemeinsame Nutzung des IP-Adress- und Portraums

Da die Container eines Pods alle im selben Netzwerk-Namespace ausgeführt werden, haben sie auch einen gemeinsamen IP-Adress- und Portraum. Das heißt, dass die Prozesse in den Containern eines Pods darauf achten müssen, sich nicht an dieselben Portnummern zu binden, da es sonst zu Konflikten kommen kann. Das betrifft aber nur Container innerhalb eines Pods. Zwischen Containern in verschiedenen Pods kann es keine Portkonflikte geben, da jeder Pod seinen eigenen Portraum aufweist. Alle Container in einem Pod haben außerdem dieselbe Loopback-Netzwerkschnittstelle, sodass sie über `localhost` miteinander kommunizieren können.

Das lineare Podnetzwerk

Alle Pods in einem Kubernetes-Cluster befinden sich in einem einzigen gemeinsam genutzten, linearen Netzwerkadressraum (siehe Bild 3.2). Dadurch kann jeder Pod jeden anderen Pod über dessen IP-Adresse erreichen. Zwischen ihnen bestehen keine NAT-Gateways (Network Address Translation). Wenn zwei Pods untereinander Netzwerkpakete austauschen, sehen sie jeweils die IP-Adresse des anderen als die Quell-IP-Adresse des Pakets.

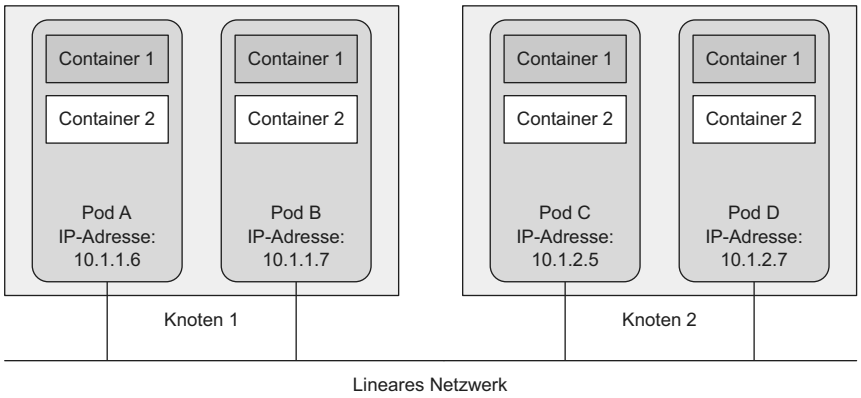


Bild 3.2 Jeder Pod erhält eine routingfähige IP-Adresse, unter der ihn die anderen Pods sehen.

Die Kommunikation zwischen den Pods ist infolgedessen sehr einfach. Unabhängig davon, ob zwei Pods auf demselben oder auf verschiedenen Arbeitsknoten ausgeführt werden, können die in ihnen enthaltenen Pods über das lineare Nicht-NAT-Netzwerk miteinander kommunizieren. Das ist damit vergleichbar, wie Computer in einem LAN unabhängig von der eigentlichen Netzwerktopologie miteinander kommunizieren. Ebenso wie solche Rechner hat jeder Pod seine eigene IP-Adresse und ist für die anderen Pods durch dieses eigens für die Pods eingerichtete Netzwerk erreichbar. Das wird gewöhnlich durch ein zusätzliches, in der Software definiertes Netzwerk erreicht, das auf das tatsächliche Netzwerk geschichtet wird.

Um den Inhalt dieses Abschnittes zusammenzufassen: Pods sind logische Hosts und verhalten sich sehr stark wie physische Hosts oder VMs. Prozesse innerhalb eines Pods ähneln den Prozessen, die auf ein und demselben physischen oder virtuellen Rechner ausgeführt werden, abgesehen davon, dass jeder Prozess in einem Container gekapselt ist.

3.1.3 Container auf Pods verteilen

Sie können sich Pods also wie eigenständige Computer vorstellen, die allerdings jeweils nur eine einzige Anwendung ausführen. Früher haben wir alle möglichen Arten von Anwendungen auf demselben Host untergebracht, aber bei Pods machen wir das nicht mehr. Da Pods recht schlank sind, können Sie so viele davon einrichten, wie Sie brauchen, ohne dabei nennenswerten Mehraufwand zu verursachen. Anstatt alles in einen einzigen Pod zu stopfen, verteilen Sie die Anwendungen auf mehrere Pods, sodass jeder nur eng miteinander in Beziehung stehende Komponenten oder Prozesse enthält.

Was glauben Sie – sollte eine mehrschichtige Anwendung, die aus einem Front-End-Anwendungsserver und einer Back-End-Datenbank besteht, in einem einzigen Pod oder in zwei Pods bereitgestellt werden?

Mehrschichtige Anwendungen auf mehrere Pods aufteilen

Es gibt zwar nichts, was Sie daran hindert, sowohl den Front-End-Server als auch die Datenbank in einem einzigen Pod mit zwei Containern auszuführen, allerdings ist es nicht die beste Vorgehensweise. Wir haben bereits gesagt, dass alle Container eines Pods immer auf demselben Knoten ausgeführt werden, aber ist es wirklich erforderlich, dass der Webserver und die Datenbank auf demselben Computer laufen? Das ist offensichtlich nicht der Fall, weshalb es nicht notwendig ist, beide in demselben Pod unterzubringen. Aber ist es auch falsch? In gewissem Sinne ist es das tatsächlich.

Wenn sich Front-End und Back-End im selben Pod befinden, laufen sie immer auf demselben Computer. Bei einem Kubernetes-Cluster mit zwei Knoten, in dem es nur diesen einen Pod gibt, wird immer nur ein einziger Arbeitsknoten verwendet. Die Rechenressourcen (CPU und Arbeitsspeicher), die auf dem zweiten Knoten zur Verfügung stehen, werden dagegen nicht genutzt. Durch die Aufteilung des Pods in zwei kann Kubernetes das Front-End auf dem einen Knoten und das Back-End auf dem anderen ausführen und dadurch die Infrastruktur besser ausnutzen.

Aufteilung aus Gründen der Skalierbarkeit

Ein weiterer Grund dafür, die beiden Schichten der Anwendung nicht in denselben Pod zu stellen, ist die Skalierung. Ein Pod ist auch das Grundelement für die Skalierung. Kubernetes kann nicht einzelne Container horizontal skalieren, sondern nur ganze Pods. Wenn Sie das Front-End und das Back-End in demselben Pod unterbringen und die Anzahl der Instanzen des Pods beispielsweise auf zwei heraufsetzen, dann haben Sie zwei Front-End- und zwei Back-End-Container.

Gewöhnlich aber besteht für Front-End-Komponenten ganz anderer Skalierungsbedarf als für Back-Ends, weshalb wir sie einzeln skalieren. Außerdem lassen sich Back-Ends wie Datenbanken gewöhnlich viel schwerer skalieren als (zustandslose) Front-End-Webserver. Wenn Sie also einen Container einzeln skalieren müssen, ist das ein deutliches Indiz dafür, dass Sie ihn in einem eigenen Pod bereitstellen sollten.

Wann Sie mehrere Container in einem Pod unterbringen sollten

Ein wichtiger Grund, um mehrere Container in einem einzigen Pod unterzubringen, liegt vor allem dann vor, wenn die Anwendung aus dem Hauptprozess und einem oder mehreren ergänzenden Prozessen besteht (siehe Bild 3.3).

Beispielsweise kann es sich bei dem Hauptcontainer in einem Pod um einen Webserver handeln, der einfach Dateien aus einem bestimmten Dateiverzeichnis zur Verfügung stellt, während ein zusätzlicher Container (auch *Sidecar* genannt, also wörtlich „Beiwagen“) regelmäßig Inhalte von einer externen Quelle herunterlädt und im Verzeichnis des Webservers speichert. In Kapitel 6 sehen wir uns an, dass wir in einem solchen Fall ein Kubernetes-*Volume* benötigen und in beiden Containern einhängen müssen.

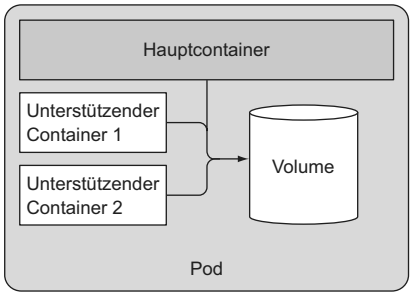


Bild 3.3 Pods sollten nur eng gekoppelte Container enthalten (gewöhnlich einen Hauptcontainer und solche, die ihn unterstützen).

Weitere Beispiele für Sidecar-Container sind Protokollrotatoren und -sammler, Datenverarbeiter, Kommunikationsadapter u. Ä.

Die Entscheidung treffen

Um zu entscheiden, ob Sie zwei Container in einen einzigen Pod stellen oder auf zwei verschiedene Pods aufteilen sollten, müssen Sie sich jeweils die folgenden Fragen stellen:

- Müssen die beiden Container zusammen ausgeführt werden oder können sie auch auf verschiedenen Hosts laufen?
- Bilden Sie ein Ganzes oder handelt es sich bei ihnen um unabhängige Komponenten?
- Müssen sie gemeinsam oder einzeln skaliert werden?

Im Allgemeinen sollten Sie Container eher in separaten Pods ausführen, sofern es keinen besonderen Grund dafür gibt, sie in einem gemeinsamen Pod unterzubringen. Bild 3.4 dient als Gedächtnisstütze.

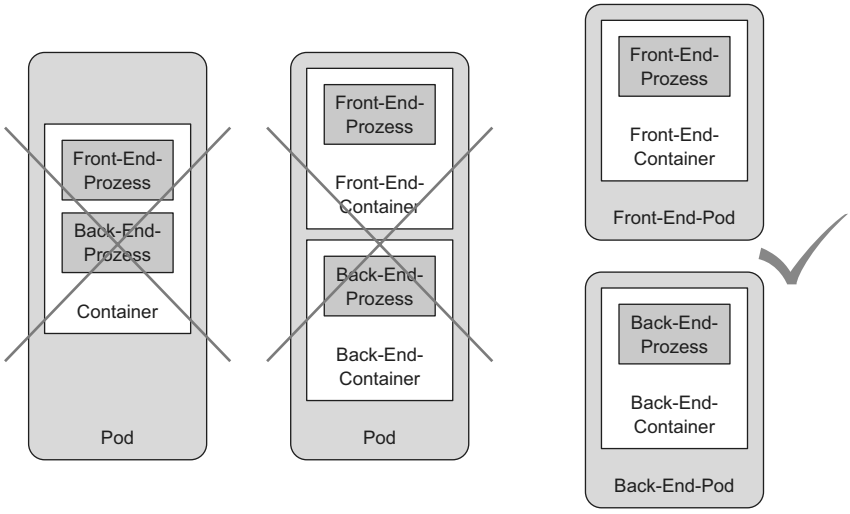


Bild 3.4 Ein Container sollte nicht mehrere Prozesse beherbergen, und ein Pod sollte nicht mehrere Container enthalten, sofern sie nicht auf demselben Computer ausgeführt werden müssen.

Pods können zwar mehrere Container enthalten, doch der Einfachheit halber wollen wir uns in diesem Kapitel nur mit Ein-Container-Pods beschäftigen. In Kapitel 6 sehen wir uns dann an, wie mehrere Container in einem Pod verwendet werden.

3.2 Pods aus YAML- und JSON-Deskriptoren erstellen

Pods und andere Kubernetes-Ressourcen erstellen Sie gewöhnlich dadurch, dass Sie ein JSON- oder YAML-Manifest am Endpunkt der REST-API von Kubernetes bereitstellen. Es gibt zwar einfachere Methoden, um solche Ressourcen anzulegen, etwa den im letzten Kapitel besprochenen Befehl `kubectl run`, allerdings können Sie dabei gewöhnlich immer nur eine Auswahl der Eigenschaften einstellen und nicht alle. Außerdem ist es bei der Definition aller Kubernetes-Objekte mit YAML-Dateien möglich, sie in einem Versionsverwaltungssystem zu speichern und all dessen Vorteile zu nutzen.

Um sämtliche Aspekte aller Arten von Ressourcen konfigurieren zu können, müssen Sie die Objektdefinitionen der Kubernetes-API kennen. Die meisten davon besprechen wir in diesem Buch zusammen mit den jeweiligen Ressourcentypen. Wir werden jedoch nicht jede einzelne Eigenschaft erklären, weshalb Sie sich zum Erstellen von Objekten auch die Dokumentation zur Kubernetes-API auf *http://kubernetes.io/docs/reference/* ansehen sollten.

3.2.1 Den YAML-Deskriptor eines bestehenden Pods untersuchen

Im vorherigen Kapitel haben wir bereits einige Pods erstellt, sodass wir uns nun ansehen können, wie die YAML-Definitionen dafür aussehen. Um diese Definition vollständig abzurufen, verwenden wir den Befehl `kubectl get` mit der Option `-o yaml`:

Listing 3.1 Vollständige YAML-Definition eines bereitgestellten Pods

```
$ kubectl get po kubia-zxzij -o yaml
apiVersion: v1   < Die Version der in diesem YAML-Deskriptor verwendeten Kubernetes-API
kind: Pod   < Typ des Kubernetes-Objekts/der Kubernetes-Ressource
metadata:                                                    | Metadaten des Pods (Name,
  annotations:                                               | Labels,
    kubernetes.io/created-by: ...                            | Anmerkungen usw.)
  creationTimestamp: 2016-03-18T12:37:50Z                    |
  generateName: kubia-                                       |
  labels:                                                    |
    run: kubia                                               |
  name: kubia-zxzij                                          |
  namespace: default                                         |
  resourceVersion: "294"                                     |
  selfLink: /api/v1/namespaces/default/pods/kubia-zxzij      |
  uid: 3a564dc0-ed06-11e5-ba3b-42010af00004                  |
```

```
spec:                                              | Spezifikation/Inhalt des Pods
  containers:                                      | (Liste der Container, Volumes
  - image: luksa/kubia                             | etc. im Pod)
    imagePullPolicy: IfNotPresent                  |
    name: kubia                                    |
    ports:                                         |
    - containerPort: 8080                          |
      protocol: TCP                                |
    resources:                                     |
      requests:                                    |
        cpu: 100m                                  |
    terminationMessagePath: /dev/termination-log   |
    volumeMounts:                                  |
    - mountPath: /var/run/secrets/k8s.io/servacc   |
      name: default-token-kvcqa                    |
      readOnly: true                               |
  dnsPolicy: ClusterFirst                          |
  nodeName: gke-kubia-e8fe08b8-node-txje           |
  restartPolicy: Always                            |
  serviceAccount: default                          |
  serviceAccountName: default                      |
  terminationGracePeriodSeconds: 30                |
  volumes:                                         |
  - name: default-token-kvcqa                      |
    secret:                                        |
      secretName: default-token-kvcqa              |
status:                                                | Ausführliche Angaben zum
  conditions:                                          | Zustand des Pods und seiner
  - lastProbeTime: null                                | Container
    lastTransitionTime: null                           |
    status: "True"                                     |
    type: Ready                                        |
  containerStatuses:                                   |
  - containerID: docker://f0276994322d247ba...         |
    image: luksa/kubia                                 |
    imageID: docker://4c325bcc6b40c110226b89fe...      |
    lastState: {}                                      |
    name: kubia                                        |
    ready: true                                        |
    restartCount: 0                                    |
    state:                                             |
      running:                                         |
        startedAt: 2016-03-18T12:46:05Z                |
  hostIP: 10.132.0.4                                   |
  phase: Running                                       |
  podIP: 10.0.2.3                                      |
  startTime: 2016-03-18T12:44:32Z                      |
```

Ich gebe zu, dass das kompliziert aussieht, allerdings wird es verständlich, wenn Sie die Grundlagen kennen und zwischen den wichtigen Teilen und den Kleinigkeiten unterscheiden können. Außerdem ist der YAML-Code, den Sie schreiben müssen, um einen neuen Pod anzulegen, viel kürzer, wie wir noch sehen werden.

Die Hauptteile der Poddefinition

Die Poddefinition besteht aus einigen wenigen Teilen. Als Erstes werden die in YAML verwendete Version der Kubernetes-API und der Typ der beschriebenen Ressource angegeben. Darauf folgen drei wichtige Abschnitte, die in fast allen Kubernetes-Ressourcen zu finden sind:

- `metadata` enthält den Namen, den Namespace, Labels sowie weitere Informationen über den Pod.
- `spec` enthält die Beschreibung der Podinhalte, also der Container, Volumes und anderen Daten.
- `status` gibt aktuelle Informationen über den laufenden Pod, darunter den Zustand des Pods, die Zustände der einzelnen Container und schließlich auch die interne IP-Adresse und andere grundlegenden Angaben.

Listing 3.1 zeigt die vollständige Beschreibung eines laufenden Pods einschließlich seines Status. Der Statusabschnitt enthält nicht schreibbare Laufzeitdaten über den augenblicklichen Zustand der Ressource. Wenn Sie einen neuen Pod erstellen, geben Sie diesen Teil niemals an.

Die drei zuvor beschriebenen Teile zeigen die typische Struktur eines Kubernetes-API-Objekts. Wie Sie in diesem Buch noch sehen werden, haben alle Objekte den gleichen Aufbau. Das erleichtert es, sich mit neuen Objekten vertraut zu machen.

Es wäre nicht sehr sinnvoll, sämtliche Eigenschaften der vorstehenden YAML-Beschreibung nacheinander durchzugehen. Stattdessen wollen wir uns den grundlegenden YAML-Code zum Erstellen eines Pods ansehen.

3.2.2 Einen einfachen YAML-Deskriptor für einen Pod schreiben

Als Nächstes erstellen wir (in einem beliebigen Verzeichnis) die Datei *kubia-manual.yaml*. Sie können auch einfach das Codearchiv zu diesem Buch herunterladen, wo Sie die Datei im Verzeichnis *Chapter03* finden. Diese Datei hat folgenden Inhalt:

Listing 3.2 Ein grundlegendes Podmanifest: *kubia-manual.yaml*

```
apiVersion: v1   < Der Deskriptor folgt Version 1 der Kubernetes-API
kind: Pod   < Wir beschreiben einen Pod
metadata:
  name: kubia-manual   < Der Name des Pods
spec:
  containers:
  - image: luksa/kubia < Das Containerimage, aus dem der Container erzeugt wird
    name: kubia   < Der Name des Containers
    ports:
    - containerPort: 8080   < Der Port, an dem die Anwendung lauscht
      protocol: TCP
```

Dies ist viel einfacher als die Definition in Listing 3.1. Sehen wir uns diesen Deskriptor im Einzelnen an. Er folgt der Version 1 der Kubernetes-API, und bei der beschriebenen Ressource handelt es sich um einen Pod namens `kubia-manual`. Dieser Pod besteht aus einem

einzelnen Container auf der Grundlage des Images `luksa/kubia`. Des Weiteren haben wir dem Container einen Namen verliehen und angegeben, dass er auf Port 8080 lauscht.

Containerports angeben

Die Angabe der Ports in der Poddefinition erfolgt rein aus informativen Gründen. Sie wegzulassen, hat keine Auswirkung darauf, ob der Pod Verbindung über diesen Port aufnehmen kann oder nicht. Wenn der Container Verbindungen über einen an die Adresse 0.0.0.0 gebundenen Port annimmt, können auch andere Pods Verbindung damit aufnehmen, selbst wenn dieser Port in ihrer Podspezifikation nicht ausdrücklich genannt wird. Es ist jedoch sinnvoll, die Ports ausdrücklich anzugeben, damit jeder, der den Cluster nutzt, auf einen Blick sehen kann, welche Ports die einzelnen Pods offenlegen. Durch die ausdrückliche Angabe der Ports können Sie ihnen auch Namen zuweisen, was sehr praktisch ist, wie wir weiter hinten in diesem Buch noch sehen werden.

Mögliche API-Objektfelder mit kubectl explain finden

Beim Schreiben eines Manifestes können Sie sich nach der Kubernetes-Dokumentation auf *kubernetes.io/docs/api* richten, um zu sehen, welche Attribute bei den verschiedenen Arten von API-Objekten angegeben werden können. Sie können dazu aber auch den Befehl `kubectl explain` verwenden.

Wenn Sie beispielsweise ein Podmanifest von Grund auf erstellen, können Sie sich von `kubectl` als Erstes eine Erklärung über Pods geben lassen:

```
$ kubectl explain pods
DESCRIPTION:
Pod is a collection of containers that can run on a host. This resource
is created by clients and scheduled onto hosts.

FIELDS:
  kind <string>
    Kind is a string value representing the REST resource this object
    represents...

  metadata <Object>
    Standard object's metadata...

  spec <Object>
    Specification of the desired behavior of the pod...

  status <Object>
    Most recently observed status of the pod. This data may not be up to
    date...
```

Kubectl gibt eine Erklärung des Objekts und eine Liste der Attribute aus, die es enthalten kann. Anschließend können Sie sich mehr über die einzelnen Attribute anzeigen lassen. Beispielsweise können wir wie folgt das Attribut `spec` untersuchen:

```
$ kubectl explain pod.spec
RESOURCE: spec <Object>

DESCRIPTION:
  Specification of the desired behavior of the pod...
  PodSpec is a description of a pod.

FIELDS:
  hostPID <boolean>
    Use the host's pid namespace. Optional: Default to false.

  ...

  volumes <[]Object>
    List of volumes that can be mounted by containers belonging to the
    pod.

  Containers <[]Object> -required-
    List of containers belonging to the pod. Containers cannot currently
    Be added or removed. There must be at least one container in a Pod.
    Cannot be updated. More info:
    http://releases.k8s.io/release-1.4/docs/user-guide/containers.md
```

3.2.3 Einen Pod mit kubectl create erstellen

Um aus unserer YAML-Datei nun einen Pod zu erstellen, verwenden wir den Befehl `kubectl create`:

```
$ kubectl create -f kubia-manual.yaml
Pod "kubia-manual" created
```

Der Befehl `kubectl create -f` dient dazu, beliebige Ressourcen (nicht nur Pods) aus einer YAML- oder JSON-Datei zu erstellen.

Die komplette Definition eines laufenden Pods abrufen

Nachdem Sie den Pod erstellt haben, können Sie Kubernetes nach seiner kompletten YAML-Definition fragen, um zu sehen, ob sie der zuvor betrachteten YAML-Definition ähnlich sieht. Was es mit den zusätzlichen Feldern in der zurückgegebenen Definition auf sich hat, besprechen wir in den folgenden Abschnitten. Zum Abrufen des vollständigen Deskriptors verwenden Sie folgenden Befehl:

```
$ kubectl get po kubia-manual -o yaml
```

Wenn Sie lieber mit JSON arbeiten, können Sie `kubectl` wie folgt auch anweisen, den JSON- statt des YAML-Deskriptors abzurufen (was auch dann funktioniert, wenn Sie den Pod mit YAML erstellt haben):

```
$ kubectl get po kubia-manual -o json
```

Der neue Pod in der Podliste

Wir haben jetzt zwar einen neuen Pod angelegt, aber woher wissen wir, dass er auch ausgeführt wird? Dazu rufen wir die Liste der Pods ab, um ihren jeweiligen Status einzusehen:

```
$ kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
kubia-manual   1/1     Running   0          32s
kubia-zxzij    1/1     Running   0          1d
```

Da haben wir unseren Pod `kubia-manual`! Laut Statusangabe wird er ausgeführt. Wenn Sie genauso veranlagt sind wie ich, möchten Sie sich vielleicht auch vergewissern, dass das wirklich stimmt, indem Sie mit dem Pod kommunizieren. Das werden wir in Kürze tun. Zunächst aber sehen wir uns das Protokoll unserer Anwendung an, um zu prüfen, ob irgendwelche Fehler aufgetreten sind.

3.2.4 Anwendungsprotokolle anzeigen

Unsere kleine Node.js-Anwendung schreibt ihre Protokolle an den Standardausgang des Prozesses. Containeranwendungen verwenden für ihre Protokolle gewöhnlich den Standardausgang und den Standardfehlerstream, anstatt sie in Dateien zu schreiben. Dadurch können die Benutzer die Protokolle der verschiedenen Anwendungen auf einfache, standardisierte Weise einsehen.

Die Containerlaufzeitumgebung (in unserem Fall Docker) leitet diese Streams in Dateien um, sodass sich das Protokoll des Containers mit folgendem Befehl abrufen lässt:

```
$ docker logs <Container-ID>
```

Sie könnten sich also mit `ssh` an dem Knoten anmelden, auf dem der Pod ausgeführt wird, und seine Protokolle mit `docker logs` abrufen, aber Kubernetes bietet eine einfachere Möglichkeit.

Das Protokoll eines Pods mit kubectl logs abrufen

Um an das Protokoll des Pods (genauer gesagt, des Containers) zu kommen, müssen wir einfach den folgenden Befehl auf dem lokalen Computer ausführen (ohne dass irgendeine `ssh`-Anmeldung erforderlich wäre):

```
$ kubectl logs kubia-manual
Kubia server starting...
```

Da wir noch keine Webanforderungen an unsere Node.js-Anwendung gesandt haben, zeigt das Protokoll nur den einen Eintrag über den Start des Servers an. Wie Sie sehen, ist es unglaublich einfach, die Protokolle einer in Kubernetes laufenden Anwendung abzurufen, wenn der Pod nur einen einzigen Container enthält.

HINWEIS: Containerprotokolle werden nach einem Tag sowie nach Erreichen der Protokolldateigröße von 10 MB zyklisch wiederverwertet. Kubect-Protokolle zeigen nur die Protokolleinträge des letzten Zyklus an.

Protokolle von Mehr-Container-Pods unter Angabe des Containernamens abrufen

Enthält der Pod mehrere Container, müssen wir beim Ausführen von `kubectl logs` mit der Option `-c <Containername>` ausdrücklich den Container angeben. Der Container in `kubia-manual` heißt `kubia`. Wenn es noch weitere Container in dem Pod gäbe, müssen wir die Protokolle von `kubia` wie folgt abrufen:

```
$ kubectl logs kubia-manual -c kubia
Kubia server starting...
```

Beachten Sie, dass Sie nur die Containerprotokolle von Pods abrufen können, die noch existieren. Wenn ein Pod gelöscht ist, dann gibt es auch seine Protokolle nicht mehr. Um die Protokolle eines Pods auch nach dessen Löschung noch verfügbar zu haben, müssen Sie eine zentrale, clusterweite Protokollierung einrichten, bei der alle Protokolle in einem zentralen Speicherbereich abgelegt werden. Wie das funktioniert, erfahren Sie in Kapitel 17.

3.2.5 Anforderungen an den Pod senden

Der Pod wird jetzt ausgefüllt – zumindest behaupten das `kubectl get` und das Protokoll unserer Anwendung. Aber wie können wir ihn uns im Einsatz ansehen? Im vorherigen Kapitel haben wir mit dem Befehl `kubectl expose` einen Dienst erstellt, um von außen Einsicht in den Pod zu nehmen. Da wir noch ein ganzes Kapitel haben werden, das Diensten gewidmet ist, wollen wir das hier nicht tun. Es gibt auch noch andere Möglichkeiten, um zum Testen und Debuggen Verbindung mit einem Pod aufzunehmen. Eine davon bietet die *Portweiterleitung*.

Einen lokalen Netzwerkport zu einem Port im Pod weiterleiten

Wenn Sie mit einem bestimmten Pod kommunizieren wollen, ohne den Weg über den Dienst zu nehmen (etwa für das Debugging oder aus anderen Gründen), können Sie eine Portweiterleitung zu dem Pod einrichten. Das erledigen Sie mit `kubectl port-forward`. Der folgende Befehl leitet den lokalen Port 8888 Ihres Computers zu unserem Pod `kubia-manual` weiter:

```
$ kubectl port-forward kubia-manual 8888:8080
... Forwarding from 127.0.0.1:8888 -> 8080
... Forwarding from [::1]:8888 -> 8080
```

Die Portweiterleitung läuft, sodass wir nun über den lokalen Port Verbindung mit unserem Pod aufnehmen können.

Über die Portweiterleitung Verbindung mit dem Pod aufnehmen

In einem anderen Terminal können wir jetzt mit `curl` eine HTTP-Anforderung über den Proxy `kubectl port-forward` auf `localhost:8888` an unseren Pod senden:

```
$ curl localhost:8888
You've hit kubia-manual
```

Bild 3.5 zeigt eine stark vereinfachte Ansicht dessen, was beim Senden dieser Anforderung geschieht. In Wirklichkeit gibt es noch einige weitere Komponenten zwischen dem `kubectl`-Prozess und dem Pod, aber sie sind für unser aktuelles Thema nicht von Bedeutung.

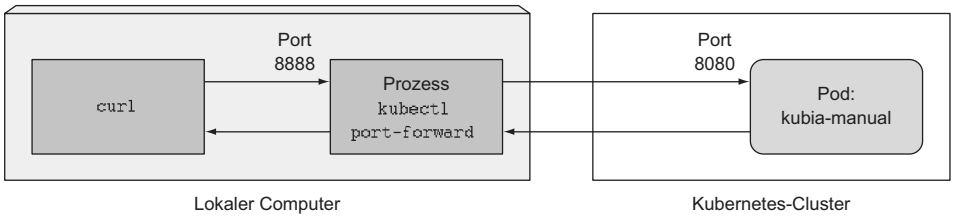


Bild 3.5 Eine stark vereinfachte Ansicht der Vorgänge bei der Verwendung von `curl` mit `kubectl port forward`

Die Verwendung der Portweiterleitung auf diese Weise ist eine wirkungsvolle Methode, um einen einzelnen Pod zu testen. In diesem Buch werden wir noch ähnliche Vorgehensweisen kennenlernen.

3.3 Pods mithilfe von Labels ordnen

Zurzeit laufen in unserem Cluster zwei Pods. Bei der Bereitstellung echter Anwendungen sind es jedoch meistens viel mehr. Je größer die Anzahl von Pods, umso wichtiger wird es, sie in Kategorien zu ordnen.

Betrachten Sie als Beispiel eine Microservice-Architektur. Dort kann die Zahl der bereitgestellten Microservices sehr schnell zwanzig oder mehr betragen. Diese Komponenten sind dann gewöhnlich auch noch repliziert (es werden mehrere Exemplare derselben Komponente bereitgestellt), und es können verschiedene Versionen (stabil, Beta, Canary usw.) gleichzeitig ausgeführt werden. Das kann dazu führen, dass sich Hunderte von Pods in unserem System tummeln. Ohne einen Ordnungsmechanismus haben wir am Ende ein riesiges, unübersichtliches Durcheinander wie in Bild 3.6. Dieses Bild zeigt die Pods mehrerer Microservices, von denen einige mehrere Replikate und einige auch unterschiedliche Versionen aufweisen.

Offensichtlich benötigen wir irgendeine Möglichkeit, um die Pods aufgrund selbst gewählter Kriterien in kleinere Gruppen zu gliedern, damit alle Entwickler und Systemadministratoren, die mit unserem System arbeiten müssen, sofort sehen können, welcher Pod wozu da ist. Außerdem wollen wir Operationen für alle Pods einer bestimmten Gruppe in einer einzigen Aktion durchführen können, anstatt diese Aktion auf jeden Pod einzeln anwenden zu müssen.

Eine solche Gliederung von Pods und anderen Kubernetes-Objekten erfolgt mithilfe von *Labels*.

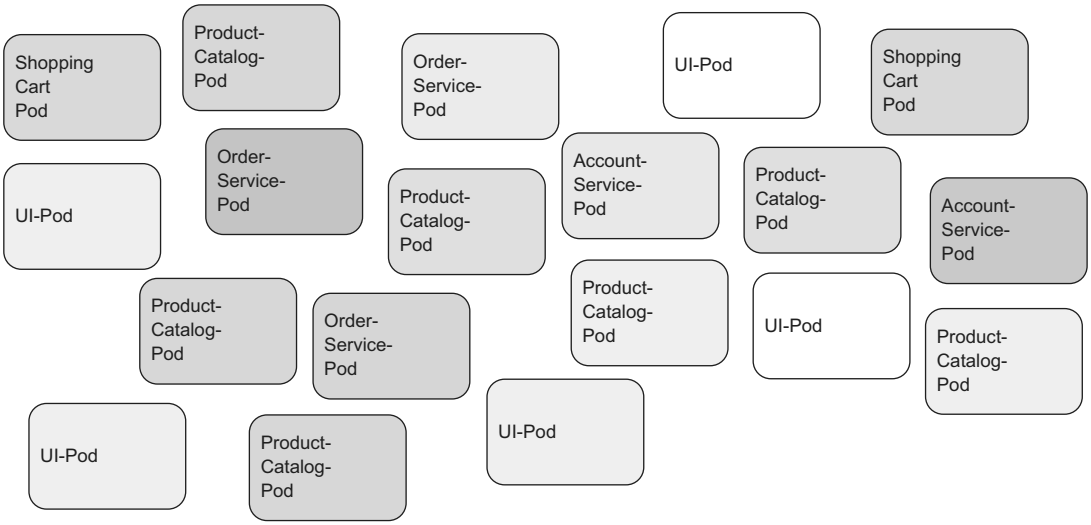


Bild 3.6 Nicht kategorisierte Pods in einer Microservice-Architektur

3.3.1 Einführung in Labels

Labels sind einfache und dort sehr vielseitige Kubernetes-Vorkehrungen, um nicht nur Pods, sondern auch alle anderen Kubernetes-Ressourcen zu ordnen. Ein Label ist ein willkürliches Schlüssel-Wert-Paar, das Sie einer Ressource zuweisen. Dieses Label wird später zur Auswahl von Ressourcen mithilfe von *Labelselektoren* herangezogen. Dabei werden die Ressourcen danach gefiltert, ob sie das in dem Selektor angegebene Label aufweisen. Eine Ressource kann mehrere Labels tragen, sofern die Schlüssel dieser Labels innerhalb der Ressource eindeutig sind. Gewöhnlich weisen Sie einer Ressource Labels zu, wenn Sie sie erstellen, aber Sie können später auch weitere Labels vergeben und sogar die Werte vorhandener Labels ändern, ohne die Ressource neu anlegen zu müssen.

Kehren wir wieder zu unserem Microservice-Beispiel aus Bild 3.6 zurück. Durch das Hinzufügen von Labels zu diesen Pods erhalten wir ein viel übersichtlicheres System, das für jeden leicht überschaubar ist. Jeder Pod wird dabei mit zwei Labels versehen:

- `app`, um anzugeben, zu welcher Anwendung oder Komponente oder welchem Microservice der Pod gehört
- `rel`, um anzugeben, ob es sich bei der Anwendung in dem Pod um ein stabiles, ein Beta- oder ein Canary-Release handelt

Definition

Bei einem Canary-Release stellen Sie eine neue Version einer Anwendung parallel zu einer stabilen Version bereit und sorgen dafür, dass nur ein Bruchteil der Benutzer zu dieser Version geleitet wird. Dadurch können Sie das

Verhalten dieser Version prüfen, bevor Sie sie für alle Benutzer bereitstellen. Diese Vorgehensweise verhindert, dass eine schlechte Version zu viele Benutzer beeinträchtigt.

Durch Hinzufügen dieser beiden Labels ordnen wir unsere Pods nach zwei Dimensionen (horizontal nach Anwendung und vertikal nach Release), wie Sie in Bild 3.7 sehen.

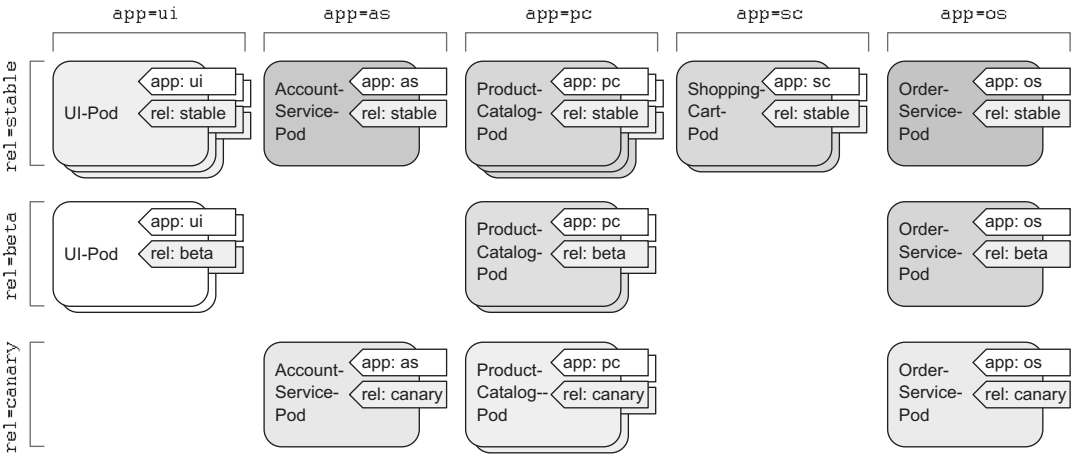


Bild 3.7 Pods in einer Microservice-Architektur mithilfe von Labels ordnen

Alle Mitglieder der Entwicklungs- und Betriebsteams, die Zugriff auf unseren Cluster haben, können jetzt leicht die Struktur des Systems erkennen und durch einen Blick auf die Labels sehen, wohin die einzelnen Pods gehören.

3.3.2 Labels beim Erstellen eines Pods angeben

Sehen wir uns das jetzt in der Praxis an, indem wir einen neuen Pod mit zwei Labels erstellen. Legen Sie dazu eine neue Datei namens *kubia-manual-with-labels.yaml* mit dem folgenden Inhalt an:

Listing 3.3 Ein Pod mit Labels: *kubia-manual-with-labels.yaml*

```
apiVersion: v1
kind: Pod
metadata:
  name: kubia-manual-v2
  labels:
    creation_method: manual   | Der Pod ist mit zwei Labels versehen
    env: prod                 |
spec:
  containers:
  - image: luksa/kubia
    name: kubia
```

```
    ports:
    - containerPort: 8080
      protocol: TCP
```

Im Abschnitt `metadata.labels` haben wir die beiden Labels `creation_method=manual` und `env=prod` angegeben. Erstellen wir nun den Pod:

```
$ kubectl create -f kubia-manual-with-labels.yaml
pod "kubia-manual-v2" created
```

Der Befehl `kubectl get pods` führt standardmäßig keine Labels auf. Sie können sie aber anzeigen lassen, indem Sie den Schalter `--show-labels` verwenden:

```
$ kubectl get po --show-labels
NAME             READY  STATUS  RESTARTS  AGE  LABELS
kubia-manual     1/1    Running  0        16m  <none>
kubia-manual-v2  1/1    Running  0        2m   creat_method=manual,env=prod
kubia-zxzij      1/1    Running  0        1d   run=kubia
```

Wenn Sie nicht alle Labels anzeigen lassen wollen, nur an einzelnen interessiert sind, können Sie sie mit dem Schalter `-L` angeben. Sie werden dann jeweils in ihrer eigenen Spalte angezeigt. Im nächsten Beispiel listen wir erneut alle Pods auf, fügen aber Spalten für die beiden an `kubia-manual-v2` angehängten Labels hinzu:

```
$ kubectl get po -L creation_method,env
NAME             READY  STATUS   RESTARTS  AGE  CREATION_METHOD  ENV
kubia-manual     1/1    Running  0         16m  <none>           <none>
kubia-manual-v2  1/1    Running  0         2m   manual           prod
kubia-zxzij      1/1    Running  0         1d   <none>           <none>
```

3.3.3 Labels vorhandener Pods ändern

Sie können auch Labels zu bestehenden Pods hinzufügen und deren Labels ändern. Da wir `kubia-manual` ebenfalls manuell erstellt haben, wollen wir ihm das Label `creation_method=manuel` hinzufügen:

```
$ kubectl label po kubia-manual creation_method=manual
pod "kubia-manual" labeled
```

Als Nächstes ändern wir das Label `env=prod` von `kubia-manual-v2` in `env=debug`, um zu zeigen, wie vorhandene Labels bearbeitet werden können.

HINWEIS: Beim Ändern vorhandener Labels müssen wir die Option `--overwrite` verwenden.

```
$ kubectl label po kubia-manual-v2 env=debug --overwrite
Pod "kubia-manual-v2" labeled
```

Wenn wir uns jetzt erneut die Liste der Pods anzeigen lassen, sehen wir die geänderten Labels:

```
$ kubectl get po -L creation_method,env
NAME             READY   STATUS    RESTARTS   AGE   CREATION_METHOD   ENV
kubia-manual     1/1     Running   0          16m   manual            <none>
kubia-manual-v2  1/1     Running   0          2m    manual            debug
kubia-zxzij      1/1     Running   0          1d    <none>            <none>
```

Wie Sie sehen, ist es ganz einfach, Labels an Ressourcen anzuhängen und zu ändern. Es mag jetzt noch nicht deutlich sein, aber dies ist eine enorm hilfreiche Möglichkeit, wie wir im nächsten Kapitel noch sehen werden. Zunächst aber wollen wir uns anschauen, was wir mit diesen Labels anstellen können, außer sie in der Podliste anzeigen zu lassen.

3.4 Eine Auswahl der Pods mithilfe von Labelselektoren auflisten

Ressourcen mit Labels zu versehen, nur um die Labels in der Ressourcenliste sehen zu können, wäre nicht sehr sinnvoll. Eine sehr praktische Verwendung haben Labels jedoch im Zusammenhang mit sogenannten *Labelselektoren*. Damit können Sie die Pods auswählen, die über ein bestimmtes Label verfügen, und eine Operation gesammelt an diesen Pods ausführen. Ein Labelselektor ist ein Kriterium, nach dem die Ressourcen gefiltert werden. Dabei ist es möglich, die Ressourcen je nachdem auszuwählen, ob sie:

- über ein bestimmtes Label (mit einem bestimmten Schlüssel) verfügen oder nicht.
- über ein Label mit einem bestimmten Schlüssel und Wert verfügen.
- über ein Label mit einem bestimmten Schlüssel, aber einem anderen als dem angegebenen Wert verfügen.

3.4.1 Pods anhand eines Labelselektors auflisten

Sehen wir uns nun die Verwendung von Labelselektoren an dem Beispiel unserer bisher erstellten Pods an. Um alle manuell erstellten Pods aufzulisten (also diejenigen mit dem Label `creation_method=manual`), gehen wir wie folgt vor:

```
$ kubectl get po -l creation_method=manual
NAME             READY   STATUS    RESTARTS   AGE
kubia-manual     1/1     Running   0          51m
kubia-manual-v2  1/1     Running   0          37m
```

Um alle Pods mit dem Label `env` unabhängig von dessen Wert aufzulisten, verwenden wir folgenden Befehl:

```
$ kubectl get po -l env
NAME             READY   STATUS    RESTARTS   AGE
kubia-manual-v2  1/1     Running   0          37m
```

Wir können auch diejenigen auflisten, die nicht über das Label `env` verfügen:

```
$ kubectl get po -l '!env'
NAME             READY   STATUS    RESTARTS   AGE
kubia-manual     1/1     Running   0          51m
kubia-zxzij      1/1     Running   0          10d
```

HINWEIS: Sie müssen `!env` in einfache Anführungszeichen stellen, damit die Bash-Shell nicht versucht, das Ausrufezeichen auszuwerten.

Wir können Pods auch mit den folgenden Labelselektoren auswählen:

- `creation_method!=manual` findet alle Pods, bei denen das Label `creation_method` einen anderen Wert als `manual` hat.
- `env in (prod,devel)` findet alle Pods, bei denen das Label `env` auf `prod` oder `devel` gesetzt ist.
- `env notin (prod,devel)` findet alle Pods, bei denen das Label `env` einen anderen Wert als `prod` oder `devel` hat.

In unserem Microservice-Beispiel können wir mit dem Labelselektor `app=pc` alle Pods auswählen, die zum Microservice für den Produktkatalog gehören (siehe Bild 3.8).

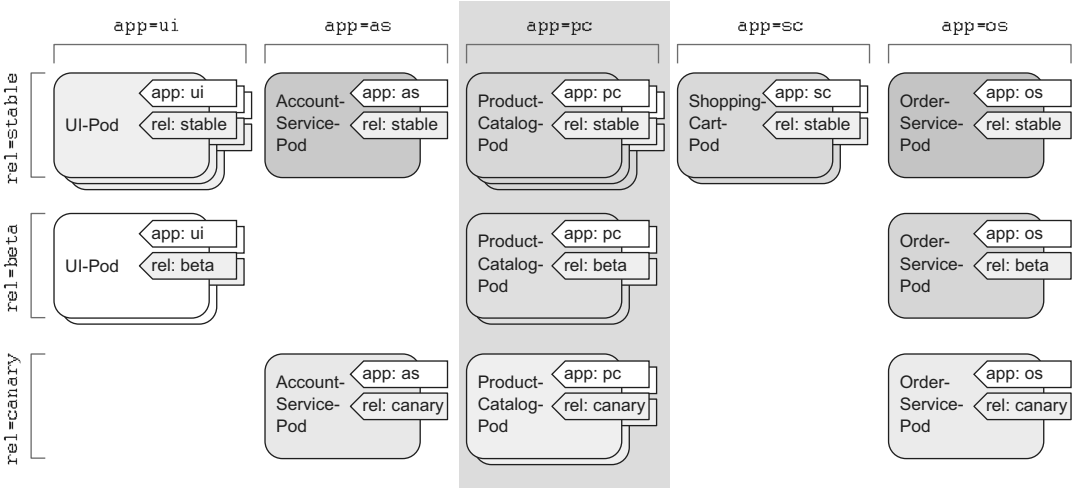


Bild 3.8 Auswahl der Pods für den Microservice des Produktkatalogs mithilfe des Labelselektors `app=pc`

3.4.2 Labelselektoren mit mehreren Bedingungen

Ein Selektor kann auch mehrere, durch Kommata getrennte Bedingungen enthalten. Um von dem Selektor ausgewählt zu werden, müssen die Ressourcen alle diese Bedingungen erfüllen. Wenn wir beispielsweise nur die Pods auswählen wollen, die das Beta-Release des Microservices für den Produktkatalog ausführen, können wir den Selektor `app=pc,rel=beta` verwenden (siehe Bild 3.9).

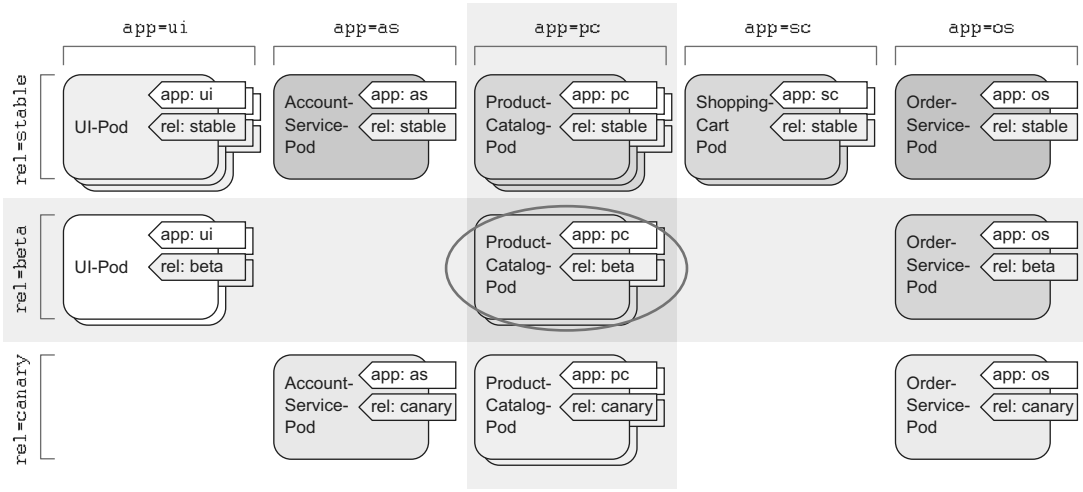


Bild 3.9 Auswählen von Pods mithilfe mehrerer Labelselektoren

Labelselektoren können wir nicht nur verwenden, um Pods für die Anzeige zu filtern, sondern auch, um Aktionen auf eine Auswahl von Pods anzuwenden. Weiter hinten in diesem Kapitel schauen wir uns beispielsweise an, wie wir damit mehrere Pods auf einmal löschen. Außerdem werden Labelselektoren nicht nur von `kubectl` genutzt, sondern auch intern verwendet, wie wir im nächsten Abschnitt sehen.

3.5 Die Podzuweisung mithilfe von Labels und Selektoren einschränken

Alle bisher von uns erstellten Pods wurden zufällig über die Arbeitsknoten verteilt. Wie ich bereits im vorherigen Kapitel gesagt habe, ist das die übliche Vorgehensweise in einem Kubernetes-Cluster. Da Kubernetes alle Knoten eines Clusters als eine riesige, geschlossene Bereitstellungsplattform exponiert, spielt es keine Rolle, welchem Knoten ein Pod zugewiesen ist. Jeder Pod bekommt genau die Menge an Rechenressourcen (CPU, Arbeitsspeicher usw.), die er angefordert hat, und auch der Zugang anderer Pods auf ihn hängt nicht davon ab, auf welchem Knoten er ausgeführt wird. Daher sollte es normalerweise nicht erforderlich sein, Kubernetes genau anzuweisen, wo es Ihre Pods zuweisen soll.

Es gibt jedoch besondere Fälle, in denen Sie zumindest ein gewisses Mitspracherecht bei dieser Zuweisung haben wollen. Ein gutes Beispiel dafür ist ein System mit inhomogener Hardware-Infrastruktur. Wenn einige der Arbeitsknoten über HDD- und andere über SSD-Laufwerke verfügen, dann kann es sein, dass Sie einige Pods lieber auf Knoten der einen Gruppe und andere lieber auf denen der anderen ausführen lassen möchten. Um ein weiteres Beispiel zu geben: Es kann auch sein, dass Sie Pods, die intensive GPU-Berechnungen vornehmen, nur Knoten zuweisen lassen möchten, die die erforderliche GPU-Beschleunigung bieten.

Sie sollten natürlich niemals genau den Knoten angeben, auf dem Sie einen Pod ausführen lassen wollen, denn dadurch würde die Anwendung an die Infrastruktur gekoppelt, was im Gegensatz zu dem Prinzip von Kubernetes steht, die Infrastruktur vor den Anwendungen zu verbergen, die darauf laufen. Wenn Sie Einfluss darauf nehmen wollen, wo ein Pod ausgeführt werden soll, geben Sie nicht konkret einen Knoten an, sondern beschreiben die Anforderungen an den Knoten und überlassen Kubernetes die Auswahl anhand dieser Anforderungen. Dazu verwenden Sie *Knotenlabels* und *Knotenlabelselektoren*.

3.5.1 Labels zur Klassifizierung von Arbeitsknoten

Wie Sie bereits wissen, sind Pods nicht die einzigen Kubernetes-Ressourcen, denen wir Labels anhängen können. Alle Kubernetes-Objekte können Labels tragen, auch Knoten. Wenn das Betriebsteam dem Cluster einen neuen Knoten hinzufügt, klassifiziert es diesen Knoten gewöhnlich, indem es ihm Labels mitgibt, die die Art der Hardware oder irgendwelche anderen Informationen angeben, die für die Zuweisung der Pods von Bedeutung sind.

Nehmen wir beispielsweise an, unser Cluster enthält einen Knoten mit einem Grafikprozessor (GPU). Um dieses Merkmal herauszustellen, fügen wir ihm wie folgt das Label `gpu=true` hinzu (wählen Sie dazu einfach einen der Knoten auf der von `kubectl get nodes` zurückgegebenen Liste aus):

```
$ kubectl label node gke-kubia-85f6-node-0rrx gpu=true
node "gke-kubia-85f6-node-0rrx" labeled
```

Jetzt können wir beim Auflisten der Knoten wie zuvor bei den Pods einen Labelselektor angeben, um nur die Knoten anzuzeigen, die über das Label `gpu=true` verfügen:

```
$ kubectl get nodes -l gpu=true
NAME                     STATUS  AGE
gke-kubia-85f6-node-0rrx  Ready   1d
```

Wie erwartet, gibt es nur einen Knoten mit diesem Label. Sie können auch versuchen, alle Knoten aufzuführen und `kubectl` anzuweisen, eine zusätzliche Spalte mit den Werten der `gpu`-Label hinzuzufügen (`kubectl get nodes -L gpu`).

3.5.2 Pods bestimmten Knoten zuweisen

Nehmen wir an, Sie wollen einen neuen Pod bereitstellen, der zur Erfüllung seiner Aufgaben auf einen Grafikprozessor zurückgreifen muss. Um den Scheduler anzuweisen, seine Auswahl auf die Knoten einzuschränken, die einen richtigen Grafikprozessor mitbringen, fügen wir der YAML-Definition des Pods einen Knotenselektor hinzu. Legen Sie dazu die Datei *kubia-gpu.yml* mit dem folgenden Inhalt an und erstellen Sie den Pod anschließend mit `kubectl create -f kubia-gpu.yaml`:

Listing 3.4 Einen Pod mithilfe eines Labelselektors einem bestimmten Knoten zuweisen: *kubia-gpu.yaml*

```
apiVersion: v1
kind: Pod
metadata:
  name: kubia-gpu
spec:
  nodeSelector:   | Dieser Knotenselektor weist Kubernetes an, diesen Pod nur
    gpu: "true"   | auf Knoten mit dem Label gpu=true bereitzustellen
  containers:
  - image: luksa/kubia
    name: kubia
```

Wir haben hier einfach im Abschnitt `spec` das Feld `nodeSelection` hinzugefügt. Wenn wir den Pod erstellen, trifft der Scheduler seine Auswahl nur unter Knoten mit dem Label `gpu=true` (das in unserem Beispiel nur ein einziger Knoten hat).

3.5.3 Zuweisung zu einem einzelnen Knoten

Da jeder Knoten über ein eindeutiges Label mit dem Schlüssel `kubernetes.io/hostname` und dem Hostnamen als Wert verfügt, können wir einen Pod auch einem einzelnen Knoten zuweisen. Wenn wir `nodeSelector` auf ein bestimmtes Hostnamenlabel setzen, kann das aber dazu führen, dass der Pod bei einem Ausfall des Knotens nicht zugewiesen werden kann. Richten Sie Ihre Überlegungen niemals auf einzelne Knoten, sondern auf logische Gruppen von Knoten, die von Knotenselektoren vorgegebene Kriterien erfüllen.

In diesem Abschnitt haben Sie einen Überblick darüber erhalten, was Labels sind, wie Labelselektoren funktionieren und wie Sie damit den Betrieb von Kubernetes beeinflussen können. Die Wichtigkeit und Nützlichkeit von Labelselektoren wird noch deutlicher, wenn wir in den nächsten beiden Kapiteln über Replikationscontroller und Dienste sprechen.

HINWEIS: Weitere Möglichkeiten, um zu beeinflussen, welchen Knoten ein Pod zugewiesen wird, lernen Sie in Kapitel 16 kennen.

3.6 Pods mit Anmerkungen versehen

Neben Labels können Pods und andere Objekte auch *Anmerkungen (Annotations)* enthalten. Dies sind ebenfalls Schlüssel-Wert-Paare, weshalb Sie ähnlich wie Labels verwendet werden. Allerdings machen sie im Gegensatz zu Labels keine Angaben, die zur Identifizierung verwendet werden können. Es ist nicht möglich, Objekte mithilfe von Anmerkungen zu gruppieren. Anmerkungsselektoren, mit denen Sie Objekte auf ähnliche Weise auswählen könnten wie mit Labelselektoren, gibt es nicht.

Dafür können Anmerkungen aber viel umfangreichere Informationen enthalten. Sie sind hauptsächlich als Werkzeuge gedacht. Manche Anmerkungen fügt Kubernetes automatisch zu Objekten hinzu, während andere manuell von den Benutzern angegeben werden müssen.

Anmerkungen sind auch bei der Einführung neuer Funktionen in Kubernetes gebräuchlich. Gewöhnlich werden in den Alpha- und Beta-Versionen neuer Merkmale keine neuen Felder zu API-Objekten hinzugefügt, sondern Anmerkungen. Wenn die erforderlichen Änderungen an der API geklärt und von allen Beteiligten abgesegnet wurden, werden die neuen Felder eingeführt und die entsprechenden Anmerkungen entfernt.

Besonders sinnvoll sind Anmerkungen, um den Pods oder anderen API-Objekten Beschreibungen hinzuzufügen, sodass jeder, der den Cluster nutzt, rasch Informationen über die einzelnen Objekte nachschlagen kann. Beispielsweise kann eine Anmerkung den Namen der Person angeben, die das Objekt erstellt hat, was die Zusammenarbeit aller Personen, die mit dem Cluster zu tun haben, stark vereinfacht.

3.6.1 Die Anmerkungen zu einem Objekt einsehen

Um uns ein Beispiel der Art von Anmerkungen anzusehen, die Kubernetes automatisch zu einem Pod hinzufügt, können wir die vollständige YAML-Definition des Pods abrufen oder den Befehl `kubectl describe` verwenden. Hier versuchen wir es zunächst mit der ersten Möglichkeit:

Listing 3.5 Die Anmerkungen zu einem Pod

```
$ kubectl get po kubia-zxzij -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubernetes.io/created-by: |
      {"kind":"SerializedReference", "apiVersion":"v1",
      "reference":{"kind":"ReplicationController", "namespace":"default", ...
```

Ohne allzu sehr ins Detail zu gehen, können Sie schon erkennen, dass die Anmerkung `kubernetes.io/created-by` JSON-Daten über das Objekt enthält, das den Pod erstellt hat. Solche Angaben wollen Sie bestimmt nicht in ein Label stellen! Labels sollen kurz sein, wohingegen Anmerkungen relativ große Datenmengen enthalten können (insgesamt bis zu 256 KB).

HINWEIS: Die Anmerkung `kubernetes.io/created-by` gilt in Version 1.8 schon als veraltet und unerwünscht und wird in Version 1.9 entfernt, weshalb Sie sie in YAML-Definitionen nicht mehr sehen werden.

3.6.2 Anmerkungen hinzufügen und ändern

Ebenso wie Labels können Anmerkungen zu Pods hinzugefügt werden, während diese erstellt werden. Es ist aber auch möglich, sie später hinzuzufügen und bereits vorhandene Anmerkungen zu ändern. Die einfachste Möglichkeit, um einem Objekt nachträglich eine Anmerkung hinzuzufügen, bietet der Befehl `kubectl annotate`.

Versuchen wir, unseren Pod `kubia-manual` um eine Anmerkung zu ergänzen:

```
$ kubectl annotate pod kubia-manual mycompany.com/someannotation="foo bar"
pod "kubia-manual" annotated
```

Hier haben wir die Anmerkung `mycompany.com/someannotation` mit dem Wert `"foo bar"` hinzugefügt. Um Konflikte zu vermeiden, sollten Sie für Anmerkungsschlüssel das hier gezeigte Format mit eindeutigen Präfixen verwenden. Wenn mehrere Werkzeuge oder Bibliotheken Anmerkungen zu Objekten machen, besteht sonst die Gefahr, dass sie gegenseitig ihre Anmerkungen überschreiben.

Um die hinzugefügte Anmerkung einzusehen, können wir `kubectl describe` versuchen:

```
$ kubectl describe pod kubia-manual | grep Annotations
...
Annotations: mycompany.com/someannotation=foo bar
...
```

3.7 Ressourcen mithilfe von Namespaces gruppieren

Kehren wir noch einmal zu den Labels zurück. Wir haben schon gesehen, wie wir Pods und andere Objekte gruppieren können. Da jedes Objekt aber mehrere Labels tragen kann, ist es möglich, dass sich Objektgruppen überlappen. Außerdem werden bei der Arbeit mit dem Cluster (z. B. mithilfe von `kubectl`) immer alle Objekte angezeigt, solange sie nicht ausdrücklich einen Labelselektor angeben.

Was aber tun Sie, wenn Sie Objekte in voneinander getrennte, nicht überlappende Gruppen aufteilen möchten, etwa um nur mit einer einzigen Gruppe zu arbeiten? Für diese und andere Zwecke gruppiert Kubernetes Objekte auch mithilfe von *Namespaces*. Dabei handelt es sich nicht um die Linux-Namespaces aus Kapitel 2, die dazu dienen, Prozesse voneinander zu isolieren. Kubernetes-Namespaces richten einfach einen Gültigkeitsbereich für

Namen ein. Anstatt alle Ressourcen in einem einzigen Namespace unterzubringen, können Sie sie auf mehrere aufteilen, sodass Sie einzelne Ressourcennamen mehrfach (nämlich in verschiedenen Namespaces) verwenden können.

3.7.1 Der Bedarf für Namespaces

Durch die Verwendung mehrerer Namespaces können Sie komplizierte Systeme mit zahlreichen Bestandteilen in kleine, scharf umrissene Gruppen zerlegen. Damit ist es beispielsweise möglich, Ressourcen in einer Mehrbenutzerumgebung aufzuteilen, z. B. in *Produktion*, *Entwicklung* und *Qualitätssicherung* oder auf jede andere sinnvolle Weise. Ressourcennamen müssen nur innerhalb eines Namespaces eindeutig sein, wohingegen in zwei verschiedenen Namespaces Ressourcen desselben Namens auftreten dürfen. Die meisten Arten von Ressourcen können einem Namespace zugeordnet werden, doch bei einigen wenigen ist das nicht der Fall. Ein Beispiel dafür sind Knoten, die global gelten und nicht an einen einzelnen Namespace gebunden sind. Weitere clusterweit gültige Ressourcen werden Sie noch in späteren Kapiteln kennenlernen.

Sehen wir uns nun an, wie Namespaces verwendet werden.

3.7.2 Andere Namespaces und die zugehörigen Pods finden

Als Erstes listen wir alle Namespaces in unserem Cluster auf:

```
$ kubectl get ns
NAME         LABELS  STATUS  AGE
default      <none>  Active  1h
kube-public  <none>  Active  1h
kube-system  <none>  Active  1h
```

Bis jetzt haben wir uns immer nur im *Standardnamespace* (`default`) bewegt. Beim Auflisten von Ressourcen mit dem Befehl `kubectl get` haben wir niemals ausdrücklich einen Namespace angegeben, weshalb `kubectl` immer auf den Standardnamespace zurückgegriffen und uns die darin enthaltenen Objekte angezeigt hat. Wie diese Liste zeigt, gibt es jedoch auch die Namespaces `kube-public` und `kube-system`. Wir können `kubectl` wie folgt anweisen, ausschließlich die Pods im Namespace `kube-system` anzuzeigen:

```
$ kubectl get po --namespace kube-system
NAME                                  READY  STATUS   RESTARTS  AGE
fluentd-cloud-kubia-e8fe-node-txje    1/1    Running  0         1h
heapster-v11-fz1ge                    1/1    Running  0         1h
kube-dns-v9-p8a4t                     0/4    Pending  0         1h
kube-ui-v4-kdlai                      1/1    Running  0         1h
l7-lb-controller-v0.5.2-bue96         2/2    Running  92        1h
```

TIPP: Statt `--namespace` können Sie auch `-n` schreiben.

Mit diesen Pods werden wir uns weiter hinten in diesem Buch noch beschäftigen. (Machen Sie sich auch keine Sorgen, wenn die hier gezeigten Pods nicht genau die gleichen sind, die Sie auf Ihrem System sehen.) Die Bezeichnung des Namespaces macht deutlich, dass diese Ressourcen zum Kubernetes-System selbst gehören. Dadurch, dass sie in einem eigenen Namespace untergebracht werden, ist alles sauber geordnet. Würden sie dagegen zusammen mit den von uns selbst erstellten Ressourcen im Standardnamespace stehen, wäre es ziemlich schwierig herauszufinden, was wozu gehört. Dabei bestünde auch die Gefahr, versehentlich Systemressourcen zu löschen.

Namespaces ermöglichen es uns, Ressourcen, die nicht zusammengehören, in nicht überlappende Gruppen aufzuteilen. Wenn mehrere Benutzer oder Benutzergruppen denselben Kubernetes-Cluster verwenden und dabei jeweils ihren eigenen Satz von Ressourcen nutzen, sollten sie auch alle ihre eigenen Namespaces haben. Dadurch besteht keine Gefahr mehr, dass sie versehentlich die Ressourcen der anderen Benutzer löschen, und sie müssen sich auch keine Gedanken mehr über Namenskonflikte machen, da die Namen ihrer Ressourcen wie bereits erwähnt nur jeweils in ihrem eigenen Namespace gültig sind.

Neben der Trennung von Ressourcen bieten Namespaces auch die Möglichkeit, den Zugriff auf manche Ressourcen auf bestimmte Benutzer einzuschränken oder die Menge der Rechenressourcen zu begrenzen, die einzelnen Benutzern zur Verfügung stehen. Mehr darüber erfahren Sie in Kapitel 12 bis 14.

3.7.3 Namespaces erstellen

Ein Namespace ist eine Kubernetes-Ressource wie jede andere, weshalb Sie sie dadurch anlegen können, dass Sie eine YAML-Datei auf den Kubernetes-API-Server stellen. Das wollen wir im Folgenden tun.

Einen Namespace mithilfe einer YAML-Datei erstellen

Erstellen Sie als Erstes die Datei *custom-namespace.yaml* mit dem folgenden Inhalt (Sie finden sie auch im Codearchiv zu diesem Buch):

Listing 3.6 YAML-Definition eines Namespaces: *custom-namespace.yaml*

```
apiVersion: v1
kind: Namespace  < Dies besagt, dass wir einen Namespace erstellen
metadata:
  name: custom-namespace  < Der Name des Namespaces
```

Anschließend platzieren Sie die Datei mithilfe von `kubectl` auf dem Kubernetes-API-Server:

```
$ kubectl create -f custom-namespace.yaml
namespace "custom-namespace" created
```

Einen Namespace mithilfe von kubectl create erstellen

Es ist zwar nicht weiter schwierig, eine Datei wie die zuvor gezeigte zu schreiben, aber trotzdem umständlich. Glücklicherweise gibt es mit `kubectl create namespace` auch einen

eigenen Befehl zum Erstellen von Namespaces, womit sich der Vorgang viel schneller erledigen lässt als mit einer YAML-Datei. Wir haben trotzdem die andere Vorgehensweise gewählt, um noch einmal deutlich zu machen, dass alle Dinge in Kubernetes API-Objekte sind, die Sie erstellen, lesen, ändern und löschen können, indem Sie ein YAML-Manifest auf den API-Server stellen.

Allerdings hätten wir den Namespace auch einfach wie folgt anlegen können:

```
$ kubectl create namespace custom-namespace
namespace "custom-namespace" created
```

HINWEIS: Die meisten Objekte müssen der in RFC 1035 aufgeführten Namenskonvention (Domänennamen) folgen, weshalb sie nur Buchstaben, Ziffern, Bindestriche und Punkte enthalten dürfen. Allerdings sind in den Namen von Namespaces (und einiger weniger anderer Ressourcen) auch keine Punkte zulässig.

3.7.4 Objekte in anderen Namespaces verwalten

Um Ressourcen in dem neuen Namespace zu erstellen, können wir Ihrer YAML-Definition das Attribut `namespace: custom-namespace` zum Feld `metadata` hinzufügen bzw. im Befehl `kubectl create` den Namespace angeben:

```
$ kubectl create -f kubia-manual.yaml -n custom-namespace
Pod "kubia-manual" created
```

Jetzt gibt es zwei Pods mit dem Namen `kubia-manual`, einen im Standardnamespace und einen in `custom-namespace`.

Um Objekte in anderen Namespaces aufzulisten, zu beschreiben, zu ändern oder zu löschen, müssen wir `kubectl` das Flag `--namespace` (oder `-n`) übergeben. Ohne die Angabe des Namespaces führt `kubectl` die Aktion in dem Standardnamespace aus, der im aktuellen `kubectl`-Kontext eingerichtet ist. Den Namespace des aktuellen Kontextes und den aktuellen Kontext selbst können Sie mithilfe der `kubectl config`-Befehle ändern. Mehr darüber erfahren Sie in Anhang A.

TIPP: Um den Standardnamespace schnell zu ändern, können Sie den Alias `alias kcd='kubectl config set-context $(kubectl config current-context) --namespace '` einrichten. Anschließend können Sie mit `kcd <Namespace>` zwischen den Namensräumen umschalten.

3.7.5 Die Trennung der Namespaces

Zum Abschluss dieses Abschnitts über Namespaces möchte ich noch erklären, was Namespaces nicht können, zumindest nicht für sich allein. Mit Namespaces können Sie Objekte in scharf abgegrenzte Gruppen aufteilen, was es Ihnen ermöglicht, Operationen gezielt auf die Mitglieder eines Namespaces anzuwenden, aber eine Isolierung laufender Objekte lässt sich damit nicht erreichen.

Nehmen wir an, verschiedene Benutzer stellen Pods jeweils in eigenen Namespaces bereit. Sind diese Pods dann voneinander isoliert und können nicht miteinander kommunizieren? Nicht notwendigerweise. Ob Namespaces eine Isolierung im Netzwerk bewirken, hängt von der verwendeten Netzwerklösung ab. Wenn diese Lösung keine Netzwerkisolierung zwischen Namespaces bietet, kann ein Pod im Namespace `foo`, der die IP-Adresse eines Pods im Namespace `bar` kennt, Datenverkehr wie beispielsweise HTTP-Anforderungen an diesen anderen Pod senden.

3.8 Pods stoppen und entfernen

Die Pods, die wir bis jetzt erstellt haben – vier im Standardnamespace und einer in `custom-namespace` –, sollten noch alle laufen. Da wir sie nicht mehr benötigen, wollen wir sie alle stoppen.

3.8.1 Pods unter Angabe des Namens löschen

Als Erstes löschen wir den Pod `kubia-gpu` unter Angabe seines Namens:

```
$ kubectl delete po kubia-gpu
pod "kubia-gpu" deleted
```

Dadurch weisen wir Kubernetes an, alle Container zu beenden, die zu dem Pod gehören. Kubernetes sendet das Signal `SIGTERM` an den Prozess und wartet eine gewisse Zeit lang (in der Standardeinstellung 30 Sekunden) darauf, dass er sauber heruntergefahren wird. Geschieht das nicht, wird der Prozess mit `SIGKILL` zwangsweise beendet. Damit die Prozesse immer sauber heruntergefahren werden, müssen daher sie das Signal `SIGTERM` korrekt handhaben.

TIPP: Sie können auch mehrere Pods auf einmal löschen, indem Sie einfach mehrere Namen durch Leerzeichen getrennt angeben (z. B. `kubectl delete po pod1 pod2`).

3.8.2 Pods mithilfe von Labelselektoren löschen

Anstatt jeden Pod, den wir löschen wollen, namentlich anzugeben, können wir auch Labelselektoren nutzen, um sowohl `kubia-manual` als auch `kubia-manual-v2` zu stoppen. Beide Pods haben das Label `creation_method=manual`, sodass wir sie wie folgt auf einen Streich löschen können:

```
$ kubectl delete po -l creation_method=manual
pod "kubia-manual" deleted
pod "kubia-manual-v2" deleted
```

In unserem Microservice-Beispiel mit Dutzenden (oder möglicherweise sogar Hunderten) von Pods können wir etwa alle Canary-Pods auf einmal löschen, indem wir den Labelselektor `rel=canary` angeben (siehe Bild 3.10):

```
$ kubectl delete po -l rel=canary
```

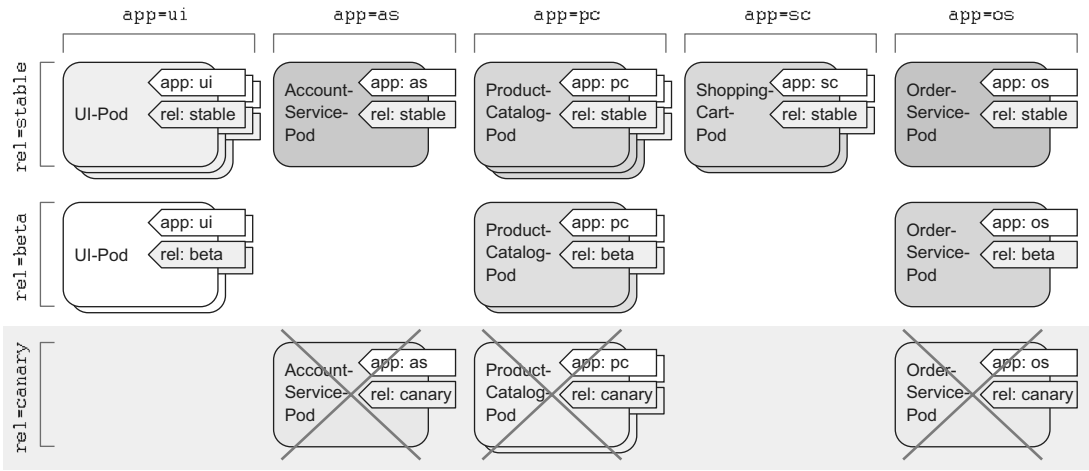


Bild 3.10 Auswählen und Löschen aller Canary-Pods mit dem Labelselektor `rel=canary`

3.8.3 Pods durch Entfernen eines ganzen Namespaces löschen

Aber nun zurück zu unserem Übungsbeispiel! Was machen wir mit dem Pod in `custom-namespace`? Da wir weder die Pods in diesem Namespace noch den Namespace selbst benötigen, können wir ihn einfach komplett löschen (wobei automatisch alle darin enthaltenen Pods entfernt werden):

```
$ kubectl delete ns custom-namespace
namespace "custom-namespace" deleted
```

3.8.4 Alle Pods in einem Namespace löschen und den Namespace erhalten

Wir haben jetzt schon fast komplett aufgeräumt. Allerdings wird der Pod, den wir in Kapitel 2 mit dem Befehl `kubectl run` erstellt haben, nach wie vor ausgeführt:

```
$ kubectl get pods
NAME          READY  STATUS   RESTARTS  AGE
kubia-zxzij   1/1    Running  0         1d
```

Anstatt gezielt diesen einen Pod zu löschen, weisen wir Kubernetes hier mit der Option `--all` an, alle Pods im aktuellen Namespace zu entfernen:

```
$ kubectl delete po --all
pod "kubia-zxzij" deleted
```

Nun wollen wir uns vergewissern, dass auch wirklich keine Pods mehr laufen:

```
$ kubectl get pods
NAME          READY  STATUS       RESTARTS  AGE
kubia-09as0   1/1    Running      0         1d
kubia-zxzij   1/1    Terminating  0         1d
```

Halt, was ist denn hier los? Wir sehen zwar, dass `kubia-zxzij` beendet wird, aber da ist ja plötzlich ein neuer Pod namens `kubia-09as0` zu sehen, der vorher nicht da war. Wie oft wir auch immer alle Pods löschen, es wird stets ein neuer Pod namens `kubia-irgendwas` auftauchen.

Denken Sie daran zurück, wie wir unseren ersten Pod mit dem Befehl `kubectl run` angelegt haben. In Kapitel 2 habe ich erwähnt, dass dadurch nicht unmittelbar ein Pod, sondern ein Replikationscontroller erstellt wird, der wiederum den Pod anlegt. Wenn wir einen Pod löschen, der von einem Replikationscontroller erstellt wurde, dann legt der Controller sofort einen neuen Pod an. Um den Pod zu entfernen, müssen wir den Replikationscontroller löschen.

3.8.5 (Fast) alle Ressourcen in einem Namespace löschen

Wir können den Replikationscontroller und alle Pods sowie alle Dienste mit einem einzigen Befehl löschen, indem wir alle Ressourcen aus dem aktuellen Namespace entfernen:

```
$ kubectl delete all --all
pod "kubia-09as0" deleted
replicationcontroller "kubia" deleted
service "kubernetes" deleted
service "kubia-http" deleted
```

Das erste `all` in diesem Befehl gibt an, dass wir alle Arten von Ressourcen löschen wollen, und mit der Option `--all` sorgen wir dafür, dass sämtliche Instanzen der Ressourcen entfernt werden, ohne dass wir sie namentlich aufführen müssen. (Diese Option haben wir schon im vorherigen `delete`-Befehl verwendet.)

HINWEIS: Auch bei der Verwendung des Schlüsselworts `all` wird nicht alles gelöscht. Einige Ressourcen (z. B. die in Kapitel 7 beschriebenen *Geheimnisse*) bleiben dabei erhalten und müssen ausdrücklich gelöscht werden.

Während des Löschvorgangs gibt `kubectl` den Namen jeder Ressource aus, die gerade entfernt wird. In der Liste sehen Sie den Replikationscontroller `kubia` sowie den Dienst `kubia-http`, die wir beide in Kapitel 2 erstellt haben.

HINWEIS: Beim Ausführen des Befehls `kubectl delete all --all` wird auch der Dienst `kubernetes` gelöscht, allerdings wird er nach wenigen Augenblicken automatisch neu erstellt.

3.9 Zusammenfassung

Nach der Lektüre dieses Kapitels sollten Sie sich jetzt mit den wichtigsten Bausteinen von Kubernetes auskennen. Jedes andere Element, das wir in den folgenden Kapiteln kennenlernen werden, steht in direktem Zusammenhang mit Pods.

In diesem Kapitel haben Sie Folgendes gelernt:

- Sie wissen, wann Sie mehrere Container in einem Pod zusammenfassen sollten und wann nicht.
- Pods können mehrere Prozesse ausführen und ähneln physischen Hosts.
- Sie können YAML- und JSON-Deskriptoren schreiben, um Pods zu erstellen, sich ihre Spezifikation anzusehen und ihren aktuellen Zustand zu untersuchen.
- Sie können Labels und Labelselektoren verwenden, um Pods zu ordnen und Operationen auf mehrere Pods auf einmal anzuwenden.
- Sie können Knotenlabels verwenden, um Pods nur Knoten zuzuweisen, die bestimmte Merkmale aufweisen.
- Mithilfe von Anmerkungen lassen sich umfangreichere Informationen zu Pods hinzufügen. Solche Anmerkungen können von Personen, aber auch von Tools und Bibliotheken erstellt werden.
- Namespaces ermöglichen es, dass mehrere Teams denselben Cluster so nutzen, als würden sie jeweils mit ihrem eigenen Cluster arbeiten.
- Mit dem Befehl `kubectl explain` können Sie Informationen über jegliche Kubernetes-Ressourcen nachschlagen.

Im nächsten Kapitel lernen Sie Replikationscontroller und andere Ressourcen zur Verwaltung von Pods kennen.

Index

Symbole

--all 92
--cascade 117
--client-certificate 583
--client-key 583
--cluster 583
--container-runtime 600
--force 335
--from-literal 223
--grace-period 335
/health 101
--it 146
--L app 112
--leader-elect 375
--namespace 87, 583
--network-plugin 370
-o custom-columns 342
--overwrite 79
--record 289
--restart=Never 328
--server 583
--sort-by 342
--token 583
--to-revision 296
--user 583
--username 583
--v 6 286
--watch 364, 483

A

Abgleichschleife 353
Abhängigkeiten
- Unterschiedliche Abhängigkeiten von Anwendungen 7

acbuild 601
Add-on
- DNS-Server 360

Add-ons
- Bereitstellen 360
- Containernetzwerk 596
- Ingress 360
- Minikube 360
- Weave Net 596

Aktivitätssonden
- Anfangsverzögerung 100
- Authentifizierung 101
- Beobachten 98
- Eigenschaften 99
- Einführung 96
- Exec-Sonde 97
- Größe 101
- Hinzufügen 97
- HTTP-Aktivitätssonde 97
- HTTP-GET-Sonde 97
- Java-Apps 101
- TCP-Socket-Sonde 97
- Wiederholungsschleifen 101
- Wirkungsvolle Sonden 100
- Zu prüfende Aspekte 100

Aktualisierung
- Ändern 292
- Änderungen einer Konfigurationszuordnung 293
- Auf eine bestimmte Revision zurücksetzen 296
- Auslösen 291
- Automatisch 279
- Bereitstellungsstrategien 290
- Bereitstellung zurücknehmen 294
- Blau-Grün-Bereitstellung 278
- Canary-Release 300
- Client- statt serverseitig 286
- Deklarativ 287
- Deployments 290
- Dienste zu neuen Pods umschalten 278

– Fehlerhaftes Rollout abbrechen 305
– Fristen für ein Rollout 305
– imagePullPolicy 282
– Imperativ 286
– kubectl 281, 283
– kubectl apply 303
– Länge des Revisionsverlaufs 297
– maxSurge 298
– maxUnavailabe 298 f.
– minReadySeconds 301
– Nachteile von kubectl rolling-update 285
– Neue Pods starten und dann alte löschen 277
– Ohne Änderung von Tags 282
– Pods erst löschen und dann ersetzen 277
– Replikationscontroller 282
– Replikationscontroller skalieren 284
– Revisionsverlauf 296
– RollingUpdate 290
– Rollout anhalten 299
– Rollout fehlerhafter Versionen verhindern 301, 304
– Rolloutrate 297
– Rollout rückgängig machen 295
– Rollout wiederaufnehmen 300
– Schrittweise 278
– Schrittweise Aktualisierung 281
– Statussätze 330
– Umschalten von alter auf neue Version 278
– Verlangsamen 291
Alias 46
– Dienste 150
allowedCapabilities 429
Amazon AWS 41
Anmerkungen
– Aktualisieren 256
– downwardAPI-Volue 256
– Einführung 85
– Einsehen 85
– Hinzufügen und ändern 86
– Mindestangaben 542
Annotations. Siehe Anmerkungen
Antiaffinitätspräferenzen 516
Antiaffinitätsregeln 514
Anwendungen
– Abhängigkeiten 7
– Aktualisieren 276
– Änderung von IP-Adresse und Hostnamen 521
– Anwendungsprotokolle 544
– Anwendungs-Templates 573
– Auf Kubernetes ausführen 22
– Automatische schrittweise Aktualisierung 279
– Automatisch neu starten 96
– Beendigung 521
– Beendigungsprozedur 533
– Bei Pods registrieren 367
– Bereitstellen 48
– Beschreibung 22
– Continuous Delivery 8
– Daten erhalten 522
– Datenmigration 534
– Deklarativ verwalten 287
– Docker 14
– Empfohlene Vorgehensweisen 546
– Entwicklung 337, 519, 546
– Grundlegender Aufbau 276
– Hochverfügbarkeit 373
– Horizontal skalieren 54
– Konfiguration ohne Neustart ändern 235
– Konfigurieren 213
– Konsistente Umgebung 8
– Kubernetes-Komponenten 519
– Leichte Handhabung 540
– Manifest 520
– Mehrschichtig 67
– Microservices 5
– Mit Statussatz bereitstellen 318
– Monolithisch 4
– Node.js 32
– Ordnungsgemäß beenden 533
– Ordnungsgemäße Verarbeitung aller Clientanforderungen 535
– Parametrisierbare Manifeste 573
– Protokolle 74
– Replikate 22
– Schrittweise Aktualisierung 278
– Umschalten von alter auf neue Version 278
– Vereinfachte Bereitstellung 25
– Vereinfachte Entwicklung 26
– Verlegung 521
– Verschwinden von Daten 522
– Zugriff 36
– Zu schnelles Herunterskalieren 314
API-Gruppe
– Benutzerdefinierte Ressourcen 556
API-Gruppen
– batch 260
– REST-Endpunkt 260
API-Server
– Adresse ermitteln 263
– Aggregation 563
– API-Dienste 563
– Aufgaben 23, 346
– Authentifizierung 258, 265
– Authentifizierungs-Plug-ins 347, 379

- Autorisierungs-Plug-in 347, 386
- Benachrichtigungen über Ressourcenänderungen 348
- Benutzer 380
- Benutzerdefiniert 562
- Benutzerdefinierte Clients 564
- Botschaftercontainer 268
- Clients 380
- Clusterverbund 604 f.
- Dienstkatalog 566
- Dienstkonten 380 f.
- Eigenen Server registrieren 563
- Einführung 22
- Gruppen 380
- Hochverfügbarkeit 375
- Identität bestätigen 264
- Kommunikation 257, 262
- Kommunikation mit Controllern 353
- Kommunikation mit etcd 343
- Kommunikation mit Pods 323
- kubectl 347
- Mehrere Instanzen 375
- Metadaten 257
- Pods zwangsweise löschen 335
- RBAC 266
- Registrieren 563
- REST-Endpunkte 259
- Serverzertifikat prüfen 264
- Sicherheit 337, 379
- URL abrufen 258
- Validierung von Objekten 346
- Verbindung während der Entwicklung 547
- Zugangssteuerungs-Plug-ins 348
- Zugriff auf interne Clusterdienste 327

API-Versionsattribut 120

Arbeitsknoten
- Akzeptable Knoten 351
- Aufgeben 492
- Auflisten 595
- Auswahl 350
- Benutzerdefinierte Mängel 500
- Besten Knoten auswählen 444
- Bester Knoten für einen Pod 351
- Bridges 368 f.
- Cluster erstellen 44
- Dateisystem 187
- Dienstunterbrechung bei Skalierung minimieren 493
- Einführung 22
- Hostport 411
- IP-Adressen ermitteln 593
- IPC-Namespace 413
- Kapazität bestimmen 444
- Kernelfunktionen 417
- Knotenaffinität 502
- Knotenausfall 109, 332
- Knotenausfall simulieren 109
- Knotencontroller 355
- Knotenprioritäten 505
- Knotenselektoren 502
- Knotenstatus 333
- Kommunikation 369
- Konfigurieren 595
- Kubernetes-Bestandteile 340
- Labels 83, 506
- Labels entfernen 126
- Labels hinzufügen 125
- Layer-3-Networking 369
- Mängel 498
- Mängel hinzufügen 500
- Manuell absperren und leeren 492
- Namensauflösung 593
- Namespace des Knotens in einem Pod verwenden 410
- Netzwerkkarte 410
- Nicht bereit 596
- Nicht-Produktionspods nicht auf Produktionsknoten ausführen 500
- NoExecute 499
- NoSchedule 499
- NotReady 333, 596
- PID-Namespace 413
- Pods bestimmten Knoten zuweisen 84
- Pods, die auf keinen Knoten passen 445
- Pods tilgen 333, 355
- Pods zuweisen 363
- PreferNoSchedule 499
- Sichtbare Prozesse 414
- Skalieren 490
- Verfügbare Ressourcen bestimmen 443
- Verfügbarkeitszonen 505
- Vollzugriff auf den Kernel 414, 417
- Wartezeit für die Neuzuteilung nach einem Knotenausfall 501
- Zugewiesene Ressourcen untersuchen 446
- Zusätzliche Knoten anfordern 491
- Zuweisbare Ressourcen 445
- Zuweisung zu einem einzelnen Knoten 84

Arbeitsspeicher
- Autoskalierung aufgrund der Arbeitsspeichernutzung 486
- Divisor 253
- Geheimnisse 246
- Grenzwert 450
- Java 453
- Nutzung auf einzelnen Knoten 469

– Nutzung in einzelnen Pods 470
– OOM-Beendigung 451
– OOMKilled 452
– Ressourcenanforderungen 442
– Ressourcenkontingent 463
– Vom Container wahrgenommener Arbeitsspeicher 453
– Wiederholtes Überschreiten des Grenzwerts 451
args 217
Atomare Aktualisierung 236
Authentifizierung
– Aktivitätssonden 101
– Als anderer Benutzer 433
– API-Server 258, 265, 379
– Benutzer 380
– Clientauthentifizierung 347
– –client-certificate 583
– –client-key 583
– Dienstkatalog 570
– Dienstkonten 380 f.
– Geheimnis 247
– Gruppen 380
– /health 101
– JSON-Webtoken 383
– –password 583
– Plug-ins 347, 379
– Registry 247
– –token 583
– Token angeben 582
– Tokendatei 381
– –username 583
automountServiceAccountToken 239
Autorisierung
– Aktionen 386
– Bearbeitung von Ressourcen 405
– Berechtigungen zum Erstellen und Ändern von Rollen 405
– Clientautorisierung 347
– Dienstkonten 382
– HTTP-Methoden 387
– Nicht-Ressourcen-URLs 398
– Plug-in 347, 386
– Prinzip der geringstmöglichen Berechtigungen 406
– RBAC 386
– RBAC-Plug-in 387
– Rechteerweiterung 405
– Reiner Lesezugriff 405
– Vollzugriff 405 f.
– Zugriff auf clusterweite Ressourcen 396
– Zugriff auf Ressourcen in allen Namespaces 401
– Zugriff auf Ressourcen in einem Namespace 400
Autoskalierer
– Benutzerdefinierte Messgröße 487
– Cluster-Autoskalierer 490, 492
– Einführung 476
– Ereignisse anzeigen 484
– Erstellen 480
– Messgrößen eines anderen Objekts 487
– Messgrößentypen 487
– minReplicas 489
– Object (Messgrößentyp) 487
– QPS 487
– Sollwert ändern 485
– Untersuchen 482
– Versionsänderungen 477
– Versionsunterschiede 482
– Vorgehensweise 481
– YAML 481
Autoskalierung
– Ablauf 476
– Andere Messgrößen 486
– Auslösen 483
– Beobachten 482
– CPU-Nutzung 479
– Deployment 484
– Erforderliche Anzahl der Pods berechnen 477
– Geeignete Messgrößen 488
– Maximale Skalierungsrate 485
– Messwerte 476
– Null Replikate 489
– Replikatanzahl ändern 478
– Speichernutzung 486
AWS 41
awsElasticBlockStore 193

B

Basisimage 33
Beendigungscodes 99
Beendigungsfrist 532
Beendigungsmeldungen 542
Beendigungspods 534
Befehle
– Argumente übergeben 215
– ENTRYPOINT/CMD 215
– Konfigurationseinträge als Befehlsargumente übergeben 228
– Shell- und Exec-Form 215
– Überschreiben 217
– Variablen initialisieren 216
Benutzer
– API-Server 380

– Bearbeiten 582
– Benutzer-ID 414
– Benutzer-ID eines Containers 415
– Gruppen 380
– Hinzufügen 582
– Konfigurationsdatei 581
– Mehrere Rollen 387
– Mit Cluster verknüpfen 582
– MustRunAs 427
– MustRunAsNonRoot 428
– Pods als anderer Benutzer erstellen 433
– Podsicherheitsrichtlinien zuweisen 431 f.
– Privilegiert 431
– Rollenbindung 393
– root 414, 416
– Zusätzliche kubectl-Benutzer 433
Benutzerdefinierte Clients 564
Benutzerdefinierte Ressourcen
– API-Version 556
– Automatisieren 558
– Benutzerdefinierte Clients 564
– Benutzerdefinierter API-Server 562
– Controlller 558
– CRD-Objekt 555
– Einführung 553
– Instanzen anzeigen 556
– Instanzen erstellen 556
– Instanzen löschen 557
– Manifest 554
– Namenskonflikte 556
– ThirdPartyResource 554
– Validieren 562
– Zweck 557
Beobachtungsmechanismus
– Controller 353
Berechtigungen
– Clusteradministrationsrechte 392
– configMap-Volume 235
– downwardAPI-Volumes 256
– Erstellen und Ändern von Rollen 405
– Kernelfähigkeiten 418
– Prinzip der geringstmöglichen Berechtigungen 406
– RBAC 387
– Rechteerweiterung 405
– secret-Volumes 244
Bereitschaftssonden
– Arten 167
– Beendigungscode 171
– Bereitschaft prüfen 169
– Einführung 167
– Einzig bereiten Pod erreichen 170
– Exec-Sonde 167
– Fehlgeschlagene Clientverbindungen beim Hochfahren verhindern 535
– Funktionsweise 167
– Hinzufügen 168
– HTTP-GET-Sonde 167
– minReadySeconds 301, 305
– Rollout einer fehlerhaften Version verhindern 302, 304
– SIGTERM 538
– TCP-Socket-Sonde 167
– Wichtigkeit 168, 171
BestEffort 454
Betriebssystem
– Installieren 587
– root 589
Bibliotheken
– Erstellen 272
– Fabric8 271
– Kommunikation 270
– Kubernetes-API 270
Blau-Grün-Bereitstellung 278
Borg 19
Botschaftercontainer
– curl 268
– Kommunikation mit API-Server 268
Bridges 368 f.
Build
– Images 34
– Manuell 35
– OpenShift 574
– Source-to-Image-Mechanismus 574
– Verzeichnis 34
Burstable 455
Busybox 30

C

cAdvisor 468, 476
Canary-Release 77, 300
Capabilities. Siehe Fähigkeiten
CAP_SYS_TIME 419
CertificateSigningRequest 165
Cgroups 13
Charts 576
CHOWN 420
CI/CD 550
CIDR-Schreibweise 438
Claim
– Persistentes Volume beanspruchen 198
Claims
– An neue Podinstanzen anhängen 316
– Auflisten 199
– Binden 199

– Einführung 195
– Erstellen 199, 315
– Löschen 315
– Ohne Angabe der Speicherklasse 208
– Pending 202
– Speicherklassen 205
– storageClassName 209
– Templates 315
– Untersuchen 206
– Verwaist 534
– Verwenden 200
– volumeClaimTemplates 321
– Vorteile 201
– Zugriffsmodi 199
– Zwangsweise binden 209
Clientbibliotheken. Siehe Bibliotheken
Clients
– Benutzerdefinierte Clients 564
– Ordnungsgemäße Verarbeitung aller Clientanforderungen 535
Cloud
– Google-Cloudprojekt einrichten 43
– Hybridcloud 603
– Ingress 161
– Zusätzliche Knoten anfordern 491
Cluster
– Anwendungen bereitstellen 48
– Architektur 21
– Auflisten 584
– Benutzer verknüpfen 582
– cluster-admin 406
– Clusteradministrationsrechte 392
– Clusterobjekt 605
– Clusterverbund 603
– Dashboard 58
– Dienste testen 139
– Dienstklassen 568
– Dienstunterbrechung bei Skalierung minimieren 493
– Eindeutige IP-Adressen 369
– Einrichten 41
– Ereignisse beobachten 364
– Erstellen 44, 585
– etcd 345, 375
– GCE Persistent Disk 190
– gcloud 44
– GKE 43
– Hinzufügen 582
– Hochverfügbarkeit 373
– Informationen über Objekte abrufen 46
– Knoten aufgeben 492
– Knoten auflisten 45
– Knotenausfall 109
– Knotenausfall simulieren 109
– Knoten skalieren 490
– Konfigurationsdatei 581
– kubeadm 585
– Löschen 584
– Mehrere Bewohner 436
– Mehrere Cluster 579
– Mehrere Knoten 585
– Mehrere Scheduler 352
– Minikube 41
– Partitionierung mit Mängeln und Tolerierungen 501
– Peer-Ermittlung 327
– Persistenter Speicher 190
– Podnetzwerk 367
– Quorum 345
– Starten 42
– tenant 436
– Überblick 44
– Überprüfen 43
– Verfügbare Dienste auflisten 568
– Vollzugriff 406
– Zentrale Protokollierung 544
– Zugriff auf clusterweite Ressourcen 396
– Zugriff auf interne Clusterdienste 327
– Zugriff vom lokalen Computer 597
Cluster-Autoskalierer 490, 492
Clusterdienstbroker 567
Clusterrollen
– admin 405
– Bearbeitung von Ressourcen 405
– cluster-admin 406
– Clusterrollenbindung 398
– edit 405
– Einführung 395
– Erstellen 396
– Manifest 396
– Nicht-Ressourcen-URLs 398
– Podsicherheitsrichtlinien 431
– Reguläre Rollenbindung 396
– Reiner Lesezugriff 405
– Standardclusterrollen 404, 406
– view 405
– Vollzugriff 405 f.
– Zugriff auf clusterweite Ressourcen 396
– Zugriff auf Ressourcen in einem Namespace 400
Clusterrollenbindungen
– Erstellen 397
– Nicht-Ressourcen-URLs 399
– Standardclusterrollenbindungen 404
– system\
  – discovery 399

– Zugriff auf Ressourcen in allen Namespaces 401
Clusterverbund
– API-Server 605
– Architektur 604
– Clusterobjekt 605
– Einführung 603
– Podliste 607
– Verbundressourcen 605 f.
– Verbundsteuerebene 604
CMD 215
CNAME 150
CNI 370
command 217
ConfigMap. Siehe Konfigurationszuordnung
Container
– Am Laufen halten 24
– Änderung der Konfigurationszuordnung 237
– Anhalten 39
– args 217
– Auflisten 36
– Auf Pods verteilen 66
– Ausführung durch Kubelet 363
– Basisimage 33
– Beendigungsmeldungen 542
– Befehle ausführen 139
– Befehle überschreiben 217
– Befehlszeilenargumente übergeben 215
– Benutzer-ID 415
– Benutzer- und Gruppen-ID 415
– Botschaftercontainer 268
– Busybox 30
– command 217
– Container Network Interface 370
– Dateien kopieren 544
– Dateisystem 38, 65
– Daten bei Neustart erhalten 522
– Debuggen 39
– default-token 239
– Definition 10
– Dienste 24
– Docker 15
– Entfernen 39
– Fähigkeiten entfernen 420, 430
– Fähigkeiten hinzufügen 419
– Fähigkeiten zu allen Containern hinzufügen 430
– Gemeinsam genutzte Volumes 179
– Gründe für die Beendigung angeben 542
– Gründe für Neustart abrufen 98
– Hello world 30
– imagePullPolicy 282
– Informationen abrufen 37
– Infrastrukturcontainer 365, 368
– Initialisierungscontainer 526
– Isolierung 13
– Kernelfähigkeiten 418
– Konfigurationseinträge übergeben 226
– Laufende Docker-Container auflisten 365
– Leicht zu handhaben 540
– Linux-Containertechnologien 10
– Mehrere Container in einem Pod 67
– Metadaten übergeben 251
– Namespaces 65
– Neustart 98
– OOMKilled 452
– Ordnungsgemäß herunterfahren 530
– Pausencontainer 365, 368
– Pods 48
– Podsicherheitsrichtlinie für privilegierte Container 431
– Ports angeben 72
– Privilegierter Modus 414, 417
– Protokolle eines abgestürzten Containers 98
– Protokoll eines abgestürzten Containers anzeigen 544
– Prozesse 64
– Prozesse auflisten 38
– Prozesse des Hostbetriebssystems 38
– QoS-Klasse 456
– Ressourcen anfordern 442
– Ressourcen einschränken 14
– root 414, 416
– runAsUser 416
– Schreiben im Dateisystem verhindern 415
– Sensible Daten 237
– Shell ausführen 37, 146
– Sicherheitskontext 414
– Sidecar 67, 186
– Standardanforderungen 460
– Standardgrenzwerte 460
– Start verzögern 526
– Teilisolierung 65
– Umgebungsvariablen 144
– Umgebungsvariablen einrichten 219
– Umgebungsvariablen in der Definition 220
– Untersuchen 37
– Unveränderlichkeit 237
– Verfügbare Ressourcen einschränken 449
– Vergleich mit VMs 11
– Verzögerter Neustart 451
– Vollzugriff auf den Kernel des Knotens 414, 417
– Volumes gemeinsam nutzen 422
– Vom Container wahrgenommene CPU-Kerne 453

– Vom Container wahrgenommener Arbeitsspeicher 453
– Zugriff bei Verlagerung auf andere Knoten 24
– Zwangsweise beenden 99
Container Network Interface 370
Container Runtime Interface 602
Continuous Delivery 8
Continuous Integration und Continuous Delivery 550
Controller
– Abgleichschleife 353
– Als Pod ausführen 560
– Aufgaben 353
– Benutzerdefinierte Ressourcen 558
– Beobachtungsereignisse 561
– Beobachtungsmechanismus 353
– Daemonsetcontroller 355
– Deployment-Controller 355, 363
– Dienstcontroller 355
– Endpunktcontroller 356
– Ereignisse beobachten 364
– Erstellen eines Replikatsatzes 363
– Hochverfügbarkeit 375
– Ingress 360
– Jobcontroller 355
– Knotencontroller 355
– Kommunikation 353
– Kooperation 361
– Namespacecontroller 356
– Persistente Volumes 356
– Pods erstellen 363
– Protokollierung 561
– Quellcode 353
– Replikationsmanager 354
– Replikatsatzcontroller 355, 363
– Statussatzcontroller 355
– Übersicht 357
– Website-Controller 558
Controller-Manager
– Benachrichtigungen über Ressourcenänderungen 348
– Clusterverbund 604
– Dienstkatalog 566
– Einführung 22
– Enthaltene Controller 352
– Hochverfügbarkeit 375
– Mehrere Instanzen 375
CPU
– Autoskalierung aufgrund der CPU-Nutzung 479
– Divisor 253
– Grenzwert überschritten 451
– Komprimierbare Ressource 449
– Nutzung auf einzelnen Knoten 469
– Nutzung in einzelnen Pods 470
– Nutzungsanzeige über 100 % 484
– Ressourcenanforderungen 442
– Ressourcenkontingent 463
– Sollwert für Autoskalierung ändern 485
– Vom Container wahrgenommene Kerne 453
– Zeitzuteilung 448
CrashLoopBackOff 543
CRD-Objekte. Siehe Benutzerdefinierte Ressourcen
CRI 602
CRI-O 602
Cron-Jobs 131
CRUD 267
curl 75
CustomResourceDefinition. Siehe Benutzerdefinierte Ressourcen

D

Daemonsets
– Controller 355
– Einen Pod auf allen Knoten ausführen 122
– Einen Pod nur auf ausgewählten Knoten ausführen 123
– Einführung 122
– Erstellen 125
– HDD/SSD-Laufwerke 124
– Labels zu Knoten hinzufügen 125
– Systemcontainer 358
– Verwendungsbeispiel 124
– Weave Net 596
– YAML-Definition 124
Dashboard 58
Dateisystem
– Arbeitsknoten 187
– Einhängen in nicht leere Verzeichnisse 233
– Isolierung 65
– Schreiben verhindern 415
– tmpfs 184
– Versteckte Dateien 233
– Zugriff 187
Datenmigration 534
Dauerhafte Speicherung. Siehe Persistenter Speicher, Volumes
Debugging
– Beendigungsmeldungen 542
– Container 39
– Gründe für die Beendigung von Containern angeben 542
– Lebenszyklushooks 529
– Zentrale Protokollierung 544

defaultAddCapabilities 429 f.
default-deny 435
default-token 238 f., 264
Deis Workflow 576
DepDeployments
– RollingUpdate 290
Deployments
– Aktualisieren 290, 303
– Alte Version 288
– Ändern 292
– Auf eine bestimmte Revision zurücksetzen 296
– Automatisch skalieren 484
– Bereitstellungsstrategien 290
– Bereitstellung zurücknehmen 294
– Clusterverbund 607
– Controller 355, 363
– Einführung 287
– Erstellen 287
– Erstellen eines Replikatsatzes 363
– Fehlerhaftes Rollout abbrechen 305
– Fristen für ein Rollout 305
– Konfiguration 574
– Konfigurationszuordnung 293
– Manifest 288
– maxSurge 298
– maxUnavailabe 298 f.
– minReadySeconds 301
– Revisionsverlauf 296
– Rollout anhalten 299
– Rollout fehlerhafter Versionen verhindern 301, 304
– Rolloutrate 297
– Rollout rückgängig machen 295
– Rollout-Status 289
– Rolloutverlauf 296
– Rollout wiederaufnehmen 300
– Vorteile 293
– Zugrunde liegender Replikationssatz 287
DevOps 8
Dienstbroker 567
Dienste
– A-Einträge 174
– Alias 150
– Anwendungen aktualisieren 278
– API-Dienste 563
– Aufgabe 53
– Auflisten 51, 390
– Außerhalb des Clusters 147
– Back-End-Dienste bei der Entwicklung 546
– Beispiel 136
– Container finden 24
– Dienstbindung 570
– Dienstbindung aufheben 571
– Dienstcontroller 355
– Dienstkatalog 564
– Dienstklassen 568
– Dienstkonten 381
– DNS 145
– Eigener Dienst für jede Podinstanz 310
– Einführung 136
– Einzig bereiten Pod erreichen 170
– Endpoints-Objekt 149, 371
– Endpunktcontroller 356
– Endpunkte 147
– Endpunktliste 147
– erstellen 51
– Erstellen 137
– ExternalName 150
– externalTrafficPolicy 158
– Externe Clients 151
– Externe Dienste 150
– Externe IP-Adresse 51
– Extern verfügbar machen 575
– Fehlerbehebung 175
– Finden 143
– Firewallregeln 153
– FQDN 145
– Headless 172
– Headless-Dienste erstellen 172
– HTTPS-Zugriff 164
– Implementierung 370
– Ingress 151, 159, 162
– Instanzen bereitstellen 569
– IP-Adresse nicht anpingbar 146
– iptables-Regeln 371
– kube-dns 360
– Kube-Proxy 358, 370
– kubernetes 93
– LoadBalancer 51, 151, 155
– Manuell eingerichtete Endpunkte 148
– Mehrere Dienste mit demselben Domänennamen 163
– Mehrere Ports 142
– Multiportdienste 142
– Netzwerk-Hops 158
– NodePort 151
– Ohne Selektor 148
– Pfade 163
– Podselektor ändern 278
– Reguläre Dienste für Haustierpods 326
– Router 575
– Sitzungsaffinität 141, 156
– SRV-Einträge 327
– Statische IP-Adresse 54
– Steuerdienst 312, 320

– Testen 139
– Umgebungsvariablen 144, 220
– Unterbrechung bei Skalierung minimieren 493
– Untersuchen 139
– Verfügbare Dienste auflisten 568
– Verschiedene Hosts 164
– Virtuelle IP-Adressen 371
– YAML-Deskriptor 138
– Zugreifen 52
– Zugriff auf interne Clusterdienste 327
– Zugriff auf Pods 51
Dienstkatalog
– Anmeldeinformationen 570
– API-Server 566
– Broker registrieren 567
– Clusterdienstbroker 567
– Controller-Manager 566
– Dienstbindung 570
– Dienstbindung aufheben 571
– Dienstbroker 567
– Dienstinstanzen bereitstellen 569
– Dienstklassen 568
– Einführung 565
– etcd 566
– Geheimnisse 570
– OpenServiceBroker 567
– Ressourcen 565
– Verfügbare Dienste auflisten 568
– Vorteile 571
Dienstkonten
– Anzeigen 381
– Autorisierung 382
– Benutzerdefiniert 384
– Benutzernamen 381
– default 381
– Dienstkonten aus anderen Namespaces in eine Rollenbindung einschließen 394
– Eigenes Konto für jeden Pod 406
– Einführung 381
– Einhängbare Geheimnisse 383
– Erstellen 382
– ImagePull-Geheimnisse 384
– JSON-Webtoken 383
– Kommunikation mit dem API-Server 385
– Mit Pods verknüpfen 384
– Rollen binden 393
– ServiceAccount 380
– Standarddienstkonto 381
– Tokendatei 381
– Token-Geheimnis 383
Dienstqualität. Siehe QoS-Klassen
dig 327
DNS 327
– Add-on 360
– Adresse des API-Servers 264
– A-Einträge 174
– Alle Pods finden 174
– CNAME 150
– Dienste finden 145
– dig 327
– ExternalName-Dienste 150
– Interner DNS-Server 145
– Lookup-Tool 327
– Namensauflösung 593
– Nicht bereite Pods finden 174
– Peer-Ermittlung einrichten 329
– Pods finden 173
– Schlüssel von Konfigurationszuordnungen 223
dnsPolicy 145
Docker
– Authentifizierung 247
– Befehle und Argumente definieren 215
– Buildvorgang 34
– CMD 215
– Container 15
– Container ausführen 30
– Daemon 34, 548
– Docker-Datei 33
– Docker Hub 30
– docker run 30
– Einführung 14
– ENTRYPOINT 215
– FROM scratch (Direktive) 540
– Grundlagen 15
– Hauptelemente 15
– Images 15
– Images auf Docker Hub hochladen 40
– Imageschichten 17
– Images erstellen und ausführen 15
– Installieren 30, 590
– Laufende Container auflisten 365
– Portierbarkeit 18
– Privates Repository 247
– Registrys 15
– Tags 35
– USER 415
– Vergleich mit VMs 16
DoesNotExist 121
Downward-API
– Begrenzte Auswahl an Metadaten 257
– downwardAPI-Volume 251, 254
– Einführung 250
– Umgebungsvariablen 251
– Verwendbare Metadaten 250
– Vorteile 257

E

echo 31
EFK 545
Egress-Regeln 438
Einweg-Pods 328
ElasticSearch 545
emptyDir 180 f.
Enable Stackdriver Logging 545
Endpoints-Objekt 148, 371, 537
Endpunktcontroller 356
Endpunktliste 147
ENTRYPOINT 215
etcd 22
– Aufbau 343
– Cluster 345, 375
– Clusterverbund 604
– Dienstkatalog 566
– Einträge für Pods 344
– Geheimnisse 345
– Hochverfügbarkeit 375
– Kommunikation 343
– Konsistenz 345
– Mehrere Instanzen 343, 345, 375
– Quorum 345
– RAFT 345
– Ressourcen speichern 343
– Schlüssel 343
– Speicherung von Geheimnissen 238
– Ungerade Anzahl von Instanzen 346
– Versionen 343
– Verwendung 343
– Verzeichnisse 343
Exec-Form 215
Exec-Sonde 97, 167
Exists 121
Extended Resources 448
ExternalName 150
externalTrafficPolicy 158

F

Fabric8 271
Fähigkeiten 418
– allowedCapabilities 429
– Bezeichnungen 419
– CAP_SYS_TIME 419
– CHOWN 420
– defaultAddCapabilities 429 f.
– Entfernen 420, 430
– Hinzufügbar 429
– Hinzufügen 419
– Podsicherheitsrichtlinien 429
– requiredDropCapabilities 429 f.
– SYS_ADMIN 429
– SYS_MODULE 429
– Zu allen Containern hinzufügen 430
FallbackToLogsOnError 543
Federation. Siehe Clusterverbund
Firewall 590
Firewallregeln 153
FluentD 545
fortune 182
fsGroup 422, 427

G

GCE Persistent Disk
– Daten schreiben 192
– Einführung 190
– Erstellen 190, 319
– gcePersistentDisk-Volume 191
– Minikube 191
– Speicherung über einen Neustart des Pods hinaus 192
gcloud 44
Geheimnisse (Secrets)
– Auflisten 239
– Authentifizierung 247
– Berechtigungen 244
– Binärdaten 242
– default-token 238, 264
– Definition 241
– Dienstkatalog 570
– Dienstkonten 383
– Einführung 238
– Einhängbar 383
– Einhängen 243
– Einträge 239
– Einträge durch Umgebungsvariablen verfügbar machen 246
– Einträge lesen 242
– Erstellen 240
– etcd 345
– Größe 242
– Identität des API-Servers 264
– ImagePull 248
– ImagePull-Geheimnisse 384
– Images abrufen 247
– JSON-Manifest 345
– Klartexteinträge 242
– Prüfen 245
– secret-Volume 239
– secret-Volumes im Arbeitsspeicher ablegen 246
– Speicherung 238

– stringData 242
– TLS 241
– TLS-Zertifikat 164
– Unterschiede zu Konfigurationszuordnungen 241
– Untersuchen 239
– Verwenden 243
gitRepo 184
GKE
– Clusteradministrationsrechte 392
– Cluster-Autoskalierer 493
– Dashboard 59
– Einführung 43
– Enable Stackdriver Logging 545
– gcloud 44
– Google-Cloudprojekt einrichten 43
– Protokollierung 545
– Umschalten zu GKE 579
Go 540
Google Cloud Monitoring 471
Google Container Registry 39
Google Kubernetes Engine. Siehe GKE
– CGE Persistent Disk 319
Grafana 471
Grafikprozessor 83
Grenzwertbereiche
– Einführung 459
– Erstellen 460
– Grenzwerte durchsetzen 461
– Kombination mit Ressourcenkontingent 464
– maxLimitRequestRatio 460
– Mindest- und Höchstwerte 460
– Plug-in 459
– Standardanforderungen 460, 462
– Standardgrenzwerte 460, 462
Gruppen
– Autorisierung 400
– Container 422
– fsGroup 422
– Gruppen-ID eines Containers 415
– Integrierte Gruppen 380
– Podsicherheitsrichtlinien zuweisen 431
– Sicherheitskontext 422
– supplementalGroups 422
– Volumes gemeinsam nutzen 422
Guaranteed 455

H

Hardware
– Bessere Ausnutzung 25
– Grafikprozessor 83, 503
– HDD/SSD-Laufwerke 83, 124
– Knotenaffinität 503
– Labels 83
– Netzwerkkarte 410
– Zuteilung von Pods zu Knoten aufgrund der Hardwarevoraussetzungen 83
Haustiere 311
Heapster 468, 476
Hello-world-Container 30
Helm 576
Höchstens-eine-Semantik 317
Hochverfügbarkeit
– Anwendungen 373
– API-Server 375
– Cluster 373
– Clusterverbund 603
– Controller 375
– etcd 375
– Inaktive Replikate 373
– Koordinatorwahl 373
– Mehrere etcd-Instanzen 343, 345
– Mehrere Instanzen 373
– Mehrere Instanzen von Komponenten der Steuerebene 342
– Scheduler 375
– Steuerebene 374
Hops 157
HorizontalPodAutoscaler. Siehe Autoskalierer
Hostname
– Änderung 521
hostNetwork 410
hostPath 188
hostPort 413
HPA. Siehe Autoskalierer
HTTP-Aktivitätssonden 97
HTTP-GET-Sonde 97, 167
HTTPS 164
Hybridcloud 603
Hypervisor 11

I

id 415
Image
– Ändern 292
ImagePull-Geheimnisse 248, 384
imagePullPolicy 282
imagePull-Richtlinie 541
Images
– Auf anderem Computer ausführen 40
– Auf Docker Hub hochladen 40
– Ausführen 31, 36
– Basisimage 33
– Betriebssystem 540

– Build-Konfiguration 574
– Buildvorgang 34
– Deployment automatisch bereitstellen 574
– Deployment-Konfiguration 574
– Docker-Daemon 548
– Docker-Datei 33
– Einführung 15
– Erstellen 33 f.
– Erstellen und ausführen 15
– FROM scratch (Direktive) 540
– Geheimnisse abrufen 247
– Go 540
– Hochladen 39
– ImagePull-Geheimnisse 248
– Leicht zu handhaben 540
– Lokal erstellen 548
– Minikube 548
– OCI-Format 601
– Ohne Änderung von Tags aktualisieren 282
– Portierbarkeit 18
– QCOW2 602
– Schichten 17, 35
– Schnelle Bereitstellung und Skalierung 540
– Tags 35, 541
– Versionsverwaltung 31
– VM-Images 602
– Zusätzliche Namen 40
ln 121
InfluxDB 471
Infrastrukturcontainer 365, 368
Ingress
– Add-on 360
– Add-on aktivieren 160
– Aktualisieren 166
– Clusterverbund 607
– Controller 360
– Dienste Pfaden zuordnen 163
– Dienste verschiedenen Hosts zuordnen 164
– Einführung 151
– Erstellen 161
– Funktionsweise 162
– Ingress-Controller 160
– IP-Adresse 162
– Mehrere Dienste mit demselben Domänennamen 163
– Minikube 160
– TLS-Datenverkehr 164
– TLS-Zertifikat 164
– Unterschiedliche Implementierungen 166
– Verwendungszweck 159
– Zugriff auf Dienste 162
– Zugriff auf Pods 162
Ingress-Regeln 434
Initialisierungscontainer 526
InitialResources 489
Inter-Process Communication. Siehe IPC
IP-Adressen
– Änderung 521
– Anpingen 146
– Anzeigen 57
– API-Server 263
– CIDR-Schreibweise 438
– Eindeutigkeit 369
– Endpoints-Objekt 148, 371
– Endpunktliste 148
– Ermitteln 593
– Extern 51
– Gemeinsame Nutzung des Adressraums 65
– Ingress 162
– iptables-Regeln 371
– JSONPath 154
– Lokal 521
– Minikube 52
– NAT 158
– Netzwerkadressübersetzung 158
– Pods 51, 66
– SNAT 158
– Statisch 54
– Verlust der Client-IP-Adresse 158
– Virtuelle Adressen für Dienste 371
IPC 64
iptables
– Hochfahren eines Pods 535
– Pod beenden 537
– Regeln 371
– userspace-Proxy 358
Isolierung
– CIDR-Schreibweise 438
– Container 10, 13
– Dateisystem 38, 65
– Namespaces 90
– Netzwerk zwischen Namespaces 436
– Podnetzwerk 434
– Prozesse 13
– Teilisolierung der Container in einem Pod 65
– VMs 10

J

Java 101
– Arbeitsspeicher 453
– Java-Client von Fabric8 271
Jobs
– API-Gruppe 260
– Ausführung geplanter Jobs 132
– Cron-Jobs 131

– Definieren 128
– Einführung 127
– Einzelne Instanzen abrufen 262
– Frist 132
– Funktionsweise 128
– Instanzen auflisten 261
– Jobcontroller 355
– Knotenausfall 127
– Mehrere Podinstanzen 129
– Pods nacheinander ausführen 129
– Pods parallel ausführen 130
– Regelmäßig ausführen 131
– Skalieren 130
– Später ausführen 131
– Template 132
– Zeitplan 132
– Zeit zum Abschließen eines Pods begrenzen 130
JSON
– Deskriptoren 73
– Ressourcendarstellung in etcd 344
– Webtoken 383
Jsonnet 549
JSONPath 154

K

Keep-alive-Verbindungen 156
Kernelfähigkeiten. Siehe Fähigkeiten
Kibana 545
Knoten
– Anzeigen 57
Knotenaffinität
– Gewichtung 507
– Hardwarevoraussetzungen für Zuteilung 503
– Knotenprioritäten 505
– Labels 506
– nodeSelectorTerms 505
– Präferenzregeln 506
– Regeln 503
– Regelnamen 504
– Standardknotenlabels 503
– Verfügbarkeitszonen 505
– Vergleich mit Knotenselektoren 502
Knotencontroller 355
Knotenmängel. Siehe Mängel
Knotenprioritäten 505
Knotenselektoren 502
Kommunikation
– API-Server 257, 262
– Ausgehenden Datenverkehr einschränken 438
– Benachrichtigungen über Ressourcenänderungen 348
– Botschaftercontainer 268
– Bridges 368 f.
– Clientbibliotheken 270
– Clientverbindungen nicht unterbrechen 535
– CNI 370
– Containernetzwerk-Plug-in 596
– Controller 353
– Dienstkatalog 567
– Dienstkontotoken 385
– Egress-Regeln 438
– Eindeutige IP-Adressen 369
– etcd 343
– Firewallregeln 153
– Infrastrukturcontainer 368
– Ingress-Regeln 434
– Inter-Process Communication 64
– IPS 64
– Kubernetes-Komponenten 341
– Layer-3-Networking 369
– Microservices 5
– –network-plugin 370
– Pausencontainer 368
– Podnetzwerk 366
– Pods 66, 323
– Pods auf verschiedenen Knoten 369
– Prozesse 64
– REST-API 5
– Software Defined Network 369
– Verbindung zum Serverpod erlauben 435
– Veth-Paar 368
Konfiguration
– Anwendungen 213
– Arbeitsknoten 595
– Atomare Aktualisierung der Dateien 236
– Authentifizierungstoken 582
– Befehle überschreiben 217
– Befehlszeilenargumente übergeben 215
– Benutzer 581
– Benutzer bearbeiten 582
– Benutzer und Cluster verknüpfen 582
– Build-Konfiguration 574
– Cluster 581
– Cluster hinzufügen 582
– Cluster löschen 584
– Deployments 574
– Einträge bearbeiten 581
– Entkoppeln 221
– Erneut laden 236
– Hartkodierung 214
– Konfigurationsdatei 580
– Konfigurationszuordnung 214, 221
– Kontext 581 f.
– Kontext löschen 584

– kubectl 580, 594
– Master 593
– Nachteile der Verwendung von Umgebungsvariablen 221
– Namespace 582
– Ohne Neustart ändern 235
– Speicherort der Konfigurationsdatei 580
– Umgebungsvariablen 214
– Umgebungsvariablen für Container 219
– Variablen mit Befehlszeilenargumenten initialisieren 216
Konfigurationszuordnung
– Alle Dateien eines Verzeichnisses importieren 224
– Alle Einträge auf einmal übergeben 227
– Ändern 237
– Atomare Aktualisierung der Dateien 236
– Bearbeiten 235
– Berechtigungen 235
– Clusterverbund 606
– configMap-Volume 229
– Dateien aktualisieren 235
– Definition 223
– Deployments 293
– Einführung 214, 221
– Einträge als Umgebungsvariablen übergeben 226
– Einzelne Einträge verfügbar machen 233
– Erneutes Laden der Konfiguration nicht zulässig 237
– Erstellen 223, 229
– Fehlende Aktualisierung von Dateien 237
– Inhalt aus Datei erstellen 224
– Inhalt eines configMap-Volumes untersuchen 232
– Konfiguration erneut laden 236
– Konfigurationsänderung ohne Neustart 235
– Konfigurationseinträge als Befehlszeilenargumente übergeben 228
– Konfigurationseinträge einhängen 234
– Konfigurationseinträge in einem Volume verwenden 231
– Konfigurationseinträge übergeben 226
– Mehrere Einträge 223
– Optionale Verweise 227
– Prüfen 232
– Schlüssel 223
– Sensible Daten 235
Konfigurationszuordnungen
– Unterschiede zu Geheimnissen 241
Konsensalgorithmus 345
Koordinatorwahl 373, 376
kops 41
Ksonnet 549
kubeadm
– Arbeitsknoten konfigurieren 595
– Cluster erstellen 585
– init 593
– join 595
– Komponenten ausführen 594
– Master konfigurieren 593
kubeclt
– Anmerkungen hinzufügen und ändern 86
– config 581
– Deployment auf eine bestimmte Revision zurücksetzen 296
– Dienste erstellen 138
– Einweg-Pods 328
– Informationen über Replikationscontroller abrufen 107
– Konfigurationseinträge bearbeiten 581
– Kontexte auflisten 584
– –restart=Never 328
– Schrittweise Aktualisierung 281
– Umschalten zu GKE 579
– Umschalten zu Minikube 579
kubectl
– Alias 46
– –all 92
– Ändern 292
– Anmerkungen einsehen 85
– annotate 86
– apply 166, 292, 303
– attach 341
– Aufgaben vor Beginn der Aktualisierung 283
– Ausführliche Protokollierung 286
– Ausgabe sortieren 342
– Befehle ausführen 139
– Benutzer bearbeiten 582
– Benutzerdefinierte Spalten 342
– Benutzer und Cluster verknüpfen 582
– Botschaftercontainer 268
– –cascade 117
– Claims auflisten 199
– –client-certificate 583
– Client des API-Servers 347
– –client-key 583
– –cluster 583
– Cluster auflisten 584
– Clusterereignisse beobachten 364
– Cluster hinzufügen 582
– cluster-info 43
– Cluster löschen 584
– cordon 492
– create 73
– Daemonsets erstellen 125

- Dateien kopieren 544
- Definition eines Replikationscontrollers ändern 115
- delete 90
- Deployment erstellen 288
- Deployments aktualisieren 303
- describe 46, 57
- Dienste untersuchen 139
- Dienstkonto erstellen 383
- Doppelter Bindestrich 140
- drain 492
- edit 114 f., 292
- exec 139
- explain 72
- expose 51, 138
- Fehlerhaftes Rollout abbrechen 305
- –force 335
- –from-literal 223
- get 45, 54
- –grace-period 335
- Informationen über Objekte abrufen 46
- Ingress aktualisieren 166
- Installieren 42
- -it 146
- Knoten auflisten 45
- Konfiguration 594
- Konfigurationsdatei 580
- Konfigurationszuordnung bearbeiten 235
- Konfigurationszuordnung erstellen 223
- Kontext 582
- Kontext löschen 584
- Kubernetes-Ressourcen ändern 292
- label 79
- -L app 112
- Laufenden Container ansprechen 341
- logs 74, 98
- Lokale Instanz 597
- Mängel anzeigen 498
- Mängel hinzufügen 500
- Master 594
- Mehrere Cluster 579
- Mehrere Ressourcen auf einmal anzeigen 483
- Nachteile von rolling-update 285
- –namespace 87, 583
- Namespace 582
- Namespace erstellen 88
- Namespace umschalten 583
- –network-plugin 370
- Objekteigenschaften ändern 291
- -o custom-columns 342
- –overwrite 79
- –password 583
- patch 291 f.
- Persistente Volumes auflisten 200
- Podselektor ändern 278
- Pods löschen 335
- port-forward 75
- Portweiterleitung 75
- Protokolle anzeigen 544
- proxy 258
- Proxyserver 258
- –record 289
- replace 292
- Replikationscontroller löschen 117
- Replikationscontroller skalieren 115
- Replikationssatz erstellen 120
- Ressourcennutzung untersuchen 442
- Revisionsverlauf 289, 296
- rolling-upate 282
- Rollout anhalten 300
- rollout history 296
- rollout pause 300
- rollout resume 300
- Rollout rückgängig machen 295
- rollout status 289
- Rollout wiederaufnehmen 300
- run 48
- scale 115
- –server 583
- set image 292
- Shell ausführen 146
- –sort-by 342
- Spalten anzeigen 112
- Speicherort der Konfigurationsdatei 580
- Status der Steuerebene prüfen 341
- Statussätze aktualisieren 330
- Statussätze erstellen 321
- Tabulatorvervollständigung 47
- taint 500
- Texteditor 115
- –token 583
- Tolerierungen anzeigen 499
- –to-revision 296
- Umschalten zwischen Kontexten 583
- uncordon 492
- undo 296
- Unterschiedliche Anzeige von Mängeln und Tolerierungen 499
- –user 583
- –username 583
- –v 6 286
- –watch 364, 483
- Zugewiesene Ressourcen untersuchen 446
- Zugriff auf API-Server 258
- Zugriff vom lokalen Computer 597
- Zusätzliche Benutzer 433

kube-dns 360
Kubelet
– Aufgaben 23, 357
– Auslagerung deaktivieren 591
– Container ausführen 363
– Einführung 22
– –network-plugin 370
– Statische Pods ausführen 357
Kube-Proxy
– Aufgaben 24
– Bezeichnung 358
– Dienste 370
– Einführung 22, 358
– Endpoints-Objekte 371
– iptables-Modus 359
– iptables-Regeln 371
– userspace-Modus 359
kube-public 87
Kubernetes
– Add-ons 340, 359
– Anwendungen ausführen 22
– Anwendungen in VMs ausführen 602
– Arbeitsknoten 21
– Architektur 339
– Automatische Skalierung 26
– Bedarf 4
– Bedeutung 4
– Bessere Ausnutzung der Hardware 25
– Cloud 26
– Cluster 21, 41
– Clusterverbund 603
– Dashboard 58
– Dienstkatalog 564
– Dienstproxy. Siehe Kube-Proxy
– Einfachere Bereitstellung von Anwendungen 25
– Einführung 4
– Entwicklung der Speicherbereitstellung 207
– Erweitern 337, 553
– Funktionsweise 361
– Geschichte 19
– Installieren 590
– Interne Mechanismen 339
– Kernaufgaben 20
– Kommunikation der Komponenten 341
– Komponenten 340
– Komponenten als Pods ausführen 342
– Labels 77
– Leicht zu handhabende Anwendungen 540
– Logische Bestandteile 52
– Master 21
– Mehrere Instanzen einer Komponente ausführen 342
– Namespaces 86
– Paketmanager 576
– Plattformen 572
– Podnetzwerk 366
– Replikate 22
– Repository hinzufügen 590
– Ressourcen ändern 292
– rkt 599
– Selbstheilung 25
– Skalierung 54
– Steuerebene 21, 340
– Systemkomponenten ausführen 358
– Überblick 19
– Verbundressourcen 605 f.
– Vereinfachte Entwicklung 26
– Verteilte Natur 340
– Vorteile 24
– Zustandsprüfung 25
KUBERNETES_SERVICE_HOST 263
KUBERNETES_SERVICE_PORT 263
kube-system 87, 594

L

Labels
– Aktualisieren 256
– Ändern 79, 112
– Arbeitsknoten klassifizieren 83
– deployment 283
– downwardAPI-Volue 256
– Einführung 77
– Entfernen 126
– Festlegen 78
– Gültigkeitsbereich eines Replikationscontrollers 111
– Hinzufügen 111
– Knotenaffinität 506
– Labelselektoren 77
– Mehrdimensional 541
– Mögliche Angaben in Labels 541
– Podzuweisung einschränken 82
– Standardknotenlabels 503
– tenant 436
– Zu Knoten hinzufügen 125
– Zuweisung zu einem einzelnen Knoten 84
Labelselektoren
– Ändern 113
– DoesNotExist 121
– Einführung 77
– Exists 121
– In 121
– Mehrere Bedingungen 82
– NotIn 121

– Operatoren 121
– Pods auswählen 80
– Pods löschen 91
– Ports 142
– Replikationscontroller 105, 113
– Replikationssätze 121
latest 35, 282
Layer-3-Networking 369
LeastRequestedPriority 444
Lebenszyklushooks
– Debugging 529
– Einführung 527
– Gültigkeitsbereich 531
– Post-Start-Hooks 528
– Pre-Stop-Hooks 530, 539
– Protokollierung 529
– SIGTERM 531
– Verbindungsunterbrechungen verhindern 539
– Verzögerung beim Herunterfahren 539
Leistung
– Imageschichten 17
– Mehrere etcd-Instanzen 343
LimitRange. Siehe Grenzwertbereich
LimitRanger 459
Linux
– Cgroups 13
– Containertechnologien 10
– /dev 417
– Gerätedateienverzeichnis 417
– Infrastrukturcontainer für Linux-Namespaces 365
– Kernelfähigkeiten 418
– Namespaces 13
– SELinux 415, 590
LoadBalancer 51
– Erstellen 155
– Minikube 52
– Verbindung aufnehmen 155

M

Mängel
– Anzeigen 498
– Aufbau 498
– Auswirkungen 499
– Benutzerdefiniert 500
– Clusterplanung 501
– Hinzufügen 500
– Nicht-Produktionspods nicht auf Produktionsknoten ausführen 500
– NoExecute 499
– NoSchedule 499
– PreferNoSchedule 499
Master
– Einführung 22
– Initialisieren 593
– Knotenstatus 333
– Konfigurieren 593
– kubectl 594
– Speicherung von Geheimnissen 238
– VM 585
– Zuweisung regulärer Pods verhindern 498
Maximale Skalierungsrate 485
maxLimitRequestRatio 460
maxSurge 298
maxUnavailabe 298 f.
metadata 71
Metadaten
– API-Server 257
– Begrenzte Auswahl in Downward-API 257
– downwardAPI-Volume 254
– Einführung 249
– Für Prozesse verfügbar machen 250
– Umgebungsvariablen 251
– Verwendbare Metadaten 250
– Volumespezifikation 256
Microservices
– Anwendungen in Microservices aufteilen 5
– Bereitstellen 6
– Geschichte 4
– Kommunikation 5
– Labels 76
– Nachteile 6
– Skalieren 6
Millicore 252
Minikube
– Add-ons 360
– Cluster starten 42
– Dashboard 59
– Einführung 41
– Entwicklung 547
– GCE Persistent Disk 191
– Images erstellen 548
– Ingress aktivieren 160
– Installieren 42
– IP-Adressen 52
– Kombination mit echtem Cluster 548
– LoadBalancer 52
– Lokale Dateien in Container einhängen 548
– Persistente Volumes 319
– Plug-ins aktivieren 425
– RBAC aktivieren 425
– rkt 600
– Swagger aktivieren 273
– Umschalten zu Minikube 579
– Zugangssteuerungs-Plug-in aktivieren 425

Minishift 575
minReadySeconds 301
minReplicas 489
Mirantis Virtlet 602
MostRequestedPriority 444
Multiportdienste 142
MustRunAs 427
MustRunAsNonRoot 428
MYSQL_ROOT_PASSWORD 214

N

Namensauflösung 593
Namensräume. Siehe Namespaces
Namespace
– Clusterverbund 606
– Dienstkonten aus anderen Namespaces in eine Rollenbindung einschließen 394
– Erstellen 390
– Höchstzahl der Objekte 466
– Infrastrukturcontainer 365
– IPC 413
– kubectl 582
– kube-system 594
– Namespacecontroller 356
– Namespace des Knotens in einem Pod verwenden 410
– Netzwerkisolierung 435
– Netzwerknamespace 368
– Netzwerk zwischen Namespaces isolieren 436
– OpenShift 573
– Persistente Volumes 198, 396
– PID 413
– Pods 266
– Ressourcen beschränken 463
– Sichtbare Prozesse 414
– Speicherklassen 204
– Umschalten 583
– Verbindung zum Serverpod erlauben 435
– Vollzugriff 405
– Zugriff auf Ressourcen in allen Namespaces 401
– Zugriff auf Ressourcen in einem Namespace 400
Namespaces
– Alle Objekte löschen 92
– Alle Pods in einem Namespace löschen 92
– Auflisten 87
– Bedarf 87
– Container 65
– default 87
– Entfernen 91
– Erstellen 88
– IPC 65
– kube-public 87
– Kubernetes-Namespaces 86
– kube-system 87
– Linux-Namespaces. Siehe
– Mit kubectl erstellen 88
– Objekte gruppieren 86
– Objekte in anderen Namespaces verwalten 89
– PID 65
– Prozesse isolieren 13
– Standardnamespace 87
– Standardnamespace ändern 89
– Trennen 90
– UTS 13
– YAML-Datei 88
NAS 190
Netzwerk
– Anforderungen 366
– Ausgehenden Datenverkehr einschränken 438
– Bridges 368
– CIDR-Schreibweise 438
– Container Network Interface 370
– Containernetzwerk-Plug-in 596
– default-deny 435
– Egress-Regeln 438
– Funktionsweise 368
– hostNetwork 410
– Hostport 411
– Infrastrukturcontainer 368
– Ingress-Regeln 434
– Isolieren 434
– Namespace 368
– Namespace des Knotens in einem Pod verwenden 410
– Netzwerkadapter 586
– Netzwerkkarte des Hosts 410
– Netzwerkrichtlinie 435
– Netzwerk zwischen Namespaces isolieren 436
– Podnetzwerk 366, 434
– Sicherheit 434
– Verbindung zum Serverpod erlauben 435
– Weave Net 596
Netzwerkadressübersetzung 158
Netzwerk-Hops 157
Netzwerkspeicher 190
Neustartrichtlinie 451
NFS 194
Node.js 32
NodePort
– Erstellen 152

– Externer Zugriff 153
– Firewallregeln 153
– Untersuchen 152
nodeSelectorTerms 505
NoExecute 499
NoOps 8
NoSchedule 499
NotIn 121
NotReady 333

O

Omega 19
OOM-Beendigung 451
OOMKilled 452
OOM-Zähler 458
Opaque Integer Resources 448
OpenAPI 272
OpenServiceBroker 567
OpenShift 572
– Anwendungs-Templates 573
– Ausprobieren 575
– Build-Konfigurationen 574
– Deployment-Konfigurationen 574
– Dienstrouten 575
– Einführung 572
– Images automatisch bereitstellen 574
– Mehrbewohnerumgebung 573
– Minishift 575
– Zusätzliche Ressourcen 572
Optimistische Parallelitätssteuerung 343

P

Pausencontainer 365, 368
Peer-Ermittlung 327
Pending 446
Persistent Disk. Siehe GCE Persistent Disk
Persistenter Speicher
– awsElasticBlockStore 193
– Claims 195
– Claim-Templates 315
– Daten schreiben 192
– GCE Persistent Disk 190
– gcePersistentDisk-Volume 191
– Haustierpods 314
– hostPath-Volumes 188
– Mehrere Replikate mit jeweils eigenem Speicher 308
– NAS 190
– Netzwerkspeicher 190
– NFS 194
– Persistente Volumes 195
– Pods von der Speichertechnologie entkoppeln 195
– Ressourcenkontingente 465
– Speichertechnologien 194
– Speicherung über einen Neustart des Pods hinaus 192
– Statussätze 314
– Systempods mit hostPath-Volumes 188
– Zuweisung desselben Speichers zu einer neuen Podinstanz 324
Persistente Volumes
– Auflisten 200, 396
– Automatisch wiederverwenden 203
– Autorisierung 396
– Claims 195, 198
– Claims löschen 315
– Claims zwangsweise binden 209
– Claim-Templates 315
– Clusterrolle 396
– Controller 356
– Dynamisch bereitstellen 204, 209
– Eigenschaften angeben 197
– Einführung 195
– Erstellen 196, 319
– Kapazität 197
– Manuell wiederverwenden 203
– Minikube 319
– Namespace 198, 396
– Provisioner 205
– RBAC 396
– Reclaim-Richtlinie 203
– Released 202
– Speicherklassen 204
– Status 202
– Statussätze 319
– Untersuchen 206
– volumeClaimTemplates 321
– Vorteile 201
– Wiederverwenden 202 f.
– Zugriffsmodi 199
PetSets 311
Pipe-Zeichen 231
Podaffinität
– Antiaffinitätspräferenzen 516
– Antiaffinitätsregeln 514
– Berücksichtigung durch Scheduler 511
– Festlegen 509
– Geografische Region 512
– Gewichtung 514
– Präferenzen 513
– topologyKey 512
– Verfügbarkeitszonen 512
Podausfallvorgabe 493

PodDisruptionBudget. Siehe Podausfallvorgabe
Pods
– Abhängigkeiten 527
– Aktivitätssonden 96
– Akzeptable Knoten 351
– Alle Pods finden 174
– Alle Pods in einem Namespace löschen 92
– Als anderer Benutzer erstellen 433
– Anforderungen 56
– Anforderungen senden 75
– Anhand von Labelselektoren auswählen 80
– Anmerkungen 85
– Anwendungen aktualisieren 276
– Anzahl der Replikate erhöhen 55
– Auf demselben Knoten bereitstellen 509
– Auflisten 49, 74, 594
– Aus dem Gültigkeitsbereich eines Replikationscontrollers entfernen 113
– Ausgehenden Datenverkehr einschränken 438
– Auswahl des Arbeitsknotens 350
– Authentifizierung 380
– Automatische schrittweise Aktualisierung 279
– Automatische Skalierung 476
– Automatische vertikale Podskalierung 489
– Beendigungsfrist 532
– Beendigungsfrist überschreiben 532
– Beendigungsmeldung 542
– Beendigungspods 534
– Benutzerdefiniertes Dienstkonto 384
– Bereitschaftssonden 167
– Bereitschaftssonden hinzufügen 168
– BestEffort 454
– Besten Knoten auswählen 444
– Bester Knoten für einen Pod 351
– Burstable 455
– Claims an neue Podinstanzen anhängen 316
– Claims verwenden 200
– Clientverbindungen nicht unterbrechen 535
– Clusterverbund 607
– CNI 370
– Container 48, 53
– Containerports angeben 72
– Container verteilen 66
– Controller als Pods ausführen 560
– Datenmigrationspod 534
– Definition abrufen 73
– Deskriptoren 69
– Deskriptor schreiben 71
– Deskriptor untersuchen 69
– Dienste 51
– Dienstkonten 380
– Dienstkonto angeben 382
– Eigene Dienstkonten 406
– Eigener Dienst für jede Podinstanz 310
– Eigener Replikationssatz für jede Podinstanz 308
– Eindeutige IP-Adressen 369
– Einen Pod auf allen Knoten ausführen 122
– Einen Pod nur auf ausgewählten Knoten ausführen 123
– Einführung 23, 48, 63
– Einträge in etcd 344
– Einweg-Pods 328
– Einzelne Pods ermitteln 172
– Einzig bereiten Pod erreichen 170
– emptyDir-Volumes verwenden 182
– Endliche Aufgaben 126
– Entkoppeln von der Speichertechnologie 195
– Erstellen 69, 73
– Erstellen durch Replikatsatzcontroller 363
– Erweiterte Podplanung 351
– Finden 173
– Front-End- und Back-End-Pods zusammenhalten 509
– Funktionsfähig halten 96
– Geheimnisse einhängen 243
– Getrennt halten 514
– Grenzwerte überschreiten 461
– Gründe für fehlgeschlagene Zuteilung ermitteln 446
– Grundlagen 65
– Guaranteed 455
– Haustierpods ersetzen 313
– Herunterfahren 531
– Horizontal skalieren 115
– hostNetwork 410
– hostPort 413
– Hostport 411
– In den Gültigkeitsbereich eines Replikationscontrollers bringen und daraus entfernen 111
– Individueller Speicher 316
– Informationen abrufen 57
– Informationen anzeigen 49
– Initialisierungscontainer 526
– Internen DNS-Server verwenden 145
– IP-Adressen 51
– IP-Adressen anzeigen 57
– IPC-Namespace 413
– Kernelfunktionen 417
– Knoten anzeigen 57
– Knotenausfall 109, 332
– Knotenprioritäten 505
– Knoten zuweisen 363
– Kommunikation 66, 323, 366

- Kommunikation mit API-Server 262
- Kommunikation zwischen Knoten 369
- Kubernetes-Komponenten als Pods ausführen 342
- Labels 76
- Labels ändern 79, 112
- Labelselektoren mit mehreren Bedingungen 82
- Labels festlegen 78
- Labels hinzufügen 111
- Laufende Pods 365
- Layer-3-Networking 369
- Lebenszyklus 521
- Lebenszyklushooks 527
- Leerlauf 489
- Linearer Netzwerkadressraum 66
- Löschen 90, 536
- Manifeste 69
- Manuell löschen 334, 532
- Mehr als die gewünschte Anzahl 103
- Mehrere Container in einem Pod 67
- Mehrere Podinstanzen in einem Job 129
- Mehrere Replikate mit jeweils eigenem Speicher 308
- Mehrere Scheduler angeben 352
- Mehrere Verzeichnisse in einem gemeinsamen Volume 309
- Mehrschichtige Anwendungen 67
- Metadaten 250
- minReadySeconds 301
- Mit Dienstkonto verknüpfen 384
- Mit Labelselektoren löschen 91
- Mit rkt ausführen 600
- Nacheinander ausführen 129
- Namespace 266
- Netzwerkkarte des Hosts 410
- Netzwerkschnittstelle 368
- Neu bereitstellen 524
- Neustartrichtlinie 451
- Nicht bereite Pods finden 174
- Nicht-Produktionspods nicht auf Produktionsknoten ausführen 500
- Ohne Angabe eines Sicherheitskontexts 415
- Ohne YAML-Manifest 173
- Parallel ausführen 130
- Pending 49
- Persistenter Speicher 314
- PID-Namespace 413
- Podaffinitätsregeln 509
- Poddefinition 71
- Podnetzwerk 366, 434
- Pods, die auf keinen Knoten passen 445
- Podsicherheitsrichtlinien 424
- Pods in einem Namespace anzeigen 87
- Portweiterleitung 75
- Post-Start-Hooks 528
- Privilegierter Modus 417
- Protokolle abrufen 74
- QoS-Klasse ermitteln 457
- QoS-Klassen 454
- Reguläre Dienste für Haustierpods 326
- Replikationscontroller 53
- Replikationscontroller löschen 117
- Ressourcen anfordern 442
- Ressourcengrenzwerte 450
- Ressourcennutzung überwachen 468
- Schrittweise Aktualisierung 278
- Schrittweise Erstellung bei Statussätzen 321
- Sichtbare Prozesse 414
- Sidecar-Container 186
- Sitzungsaffinität 141
- Ständig abstürzende Pods 524
- Startreihenfolge 525
- Statische Pods 357
- Statusbehaftete Pods 307
- Steuerdienst 312
- Steuerebene 594
- Streuen 515
- Systempods mit hostPath-Volumes 188
- Teilisolierung der Container 65
- Template 105
- Template ändern 114
- Terminating 335
- Tilgen 333, 355
- Tokendatei 381
- Tolerierungen 499
- Tolerierungen hinzufügen 500
- Tot 524
- Unbekannter Status 333
- Unveränderliche Identität 309, 316
- Unveränderliche Netzwerkidentität 312
- Verbindungsbereitschaft signalisieren 166
- Verbindung zum Serverpod erlauben 435
- Verwendbare Metadaten 250
- Verwendungszweck 64
- Volumetypen einschränken 430
- Vorgänge beim Erstellen 361
- Wartezeit für die Neuzuteilung nach einem Knotenausfall 501
- Zeit zum Abschließen eines Pods begrenzen 130
- Zugriff 51
- Zugriff auf Kubernetes-API 267
- Zugriff über Ingress 162
- Zum Löschen markiert 335
- Zuweisung desselben Speichers 324

– Zuweisung zu Arbeitsknoten über Labels einschränken 82
– Zuweisung zu bestimmten Knoten 84
– Zuweisung zu einem einzelnen Knoten 84
– Zwangsweise löschen 335
PodSecurityPolicy 424
Podsicherheitsrichtlinien
– allowedCapabilities 429
– Benutzern und Gruppen zuweisen 431
– Benutzern zuweisen 432
– Bereitstellung privilegierter Container 431
– defaultAddCapabilities 429 f.
– Einstellungen 425
– Erzwingen 427
– Fähigkeiten 429
– Fähigkeiten entfernen 430
– Fähigkeiten zu allen Containern hinzufügen 430
– Hinzufügbare Fähigkeiten 429
– MustRunAs 427
– MustRunAsNonRoot 428
– Plug-in 424
– RBAC 432
– requiredDropCapabilities 429 f.
– RunAsAny 427
– Untersuchen 426
– Volumetypen einschränken 430
– Zuweisung 432
Ports
– Angeben 72
– API-Server 263
– Benannt 142
– Dienste mit mehreren Ports 142
– Gemeinsame Nutzung des Portraums 65
– Hostport 411
– NodePort 151
– Portweiterleitung 75
Post-Start-Hooks 528
PreferNoSchedule 499
Pre-Stop-Hooks 530, 539
Prinzip der geringstmöglichen Berechtigungen 406
ProgressDeadlineExceeded 305
Protokolle
– Abgestürzte Container 98
– Abrufen 74
– Anwendungsprotokolle 544
– Ausführliche Protokollierung von kubectl 286
– Beobachtungsereignisse 561
– Clusterweite Protokollierung 544
– Controller 561
– Debuginformationen 542
– ElasticSearch 545
– Enable Stackdriver Logging 545
– GKE 545
– Jiouereb 544
– Lebenszyklushooks 529
– Mehr-Container-Pods 75
– Mehrzeilige Protokolleinträge 545
– Standardausgabe 74
– –v 6 286
– Wiederverwertung 74
– Zentrale Protokollierung 544
Protokollierung
– EFK 545
– FluentD 545
Provisioner 205
Prozesse
– Beenden 457
– Beendigungsmeldungen 542
– Beendigungsreihenfolge 457
– Benutzer-ID 414
– Container 64
– Gründe für die Beendigung angeben 542
– Hostbetriebssystem 38
– Inter-Process Communication 64
– IPC 64
– Isolierung 13
– Kommunikation 64
– Metadaten verfügbar machen 250
– Namespaces 13
– OOM-Beendigung 451
– OOM-Zähler 458
– Prozesse eines Containers auflisten 38
– Ressourcen einschränken 14
– root 416
– Schreiben im Dateisystem verhindern 415
– Sichtbar 414
– Zu beendende Prozesse auswählen 457

Q

QCOW2 602
QoS-Klassen
– Ausgabe 457
– Beendigungsreihenfolge 457
– BestEffort 454
– Beziehung zu Ressourcenanforderungen und -grenzwerten 456
– Container 456
– Container derselben Klasse 458
– Ermitteln 456
– Festlegen 454
– Guaranteed 455
– Pods 457

– Ressourcenkontingente 467
– Vorhandene Klassen 454
– Zu beendende Prozesse auswählen 457
QoS-Klasssen
– Burstable 455
QPS 477, 487
Quality of Service. Siehe QoS-Klassen
Quay.io 39
Quorum 345

R

RAFT 345
RBAC
– Aktionen 386
– Aktivieren 425
– Ausschalten 266
– Autorisierungs-Plug-in 386
– Bearbeitung von Ressourcen 405
– Berechtigungen 387
– Berechtigungen zum Erstellen und Ändern von Rollen 405
– Clusterrollen 395
– Clusterrollenbindungen 397
– Clusterrollen mit regulärer Rollenbindung 396
– Dienstkonten aus anderen Namespaces in eine Rollenbindung einschließen 394
– Kombinationen der Rollen- und Bindungstypen 403
– Mehrere Rollen 387
– Minikube 425
– Nicht-Ressourcen-URLs 398
– Plug-in 387
– Podsicherheitsrichtlinien zuweisen 431 f.
– Rechteerweiterung 405
– Reiner Lesezugriff 405
– Ressourcen 388
– Rolle erstellen 392
– Rollen 391
– Rollenbindung 393
– Standardclusterrollen 404, 406
– Standardclusterrollenbindungen 404
– Vollzugriff 405 f.
– Zugriff auf clusterweite Ressourcen 396
– Zugriff auf Ressourcen in allen Namespaces 401
– Zugriff auf Ressourcen in einem Namespace 400
readOnlyRootFilesystem 422
Rechteerweiterung 405
Reclaim-Richtlinie 203
Red Hat OpenShift 572
Registrys
– Authentifizierung 247
– Docker Hub 30
– Einführung 15
– Images auf Docker Hub hochladen 40
– Images hochladen 39
ReplicaSet. Siehe Replikationssatz
Replikate
– Anzahl ändern 24
– Anzahl erhöhen 55
– Einführung 22
– Inaktive Replikate 373
– maxSurge 298
– maxUnavailabe 298 f.
– Mehrere Replikate mit jeweils eigenem Speicher 308
– Mehrere Verzeichnisse in einem gemeinsamen Volume 309
– Null 489
– Pods 23
– Pods mit Hostport 412
– Skalierung 478
– Statusbehaftete Pods 307
– Unveränderte Anzahl bei Aktualisierung eines Deployments 303
Replikationscontroller
– Abgleichschleife 104
– Anzahl der Replikate erhöhen 55
– Aufgabe 53
– Auflisten 54
– Automatische schrittweise Aktualisierung 279
– Beobachten 106
– Definition ändern 115
– Einführung 102
– Ersatzpods 102
– Erstellen 105
– Funktionsweise 103
– Gültigkeitsbereich 111
– Hauptbestandteile 104
– Herunterskalieren 116
– Hochskalieren 115
– Informationen abrufen 107
– Labelselektor 105
– Labelselektor ändern 113
– Löschen 117
– Mehr als die gewünschte Anzahl von Pods 103
– Pods aus Gültigkeitsbereich entfernen 113
– Podselektor 106
– Pods erstellen 53
– Pod-Template 105
– Pod-Template ändern 114

– Reaktion auf Knotenausfall 109
– Reaktion auf Löschung eines Pods 107
– Replikationsmanager 354
– Schrittweise Aktualisierung 282
– Skalieren zur Aktualisierung 284
– Verfügbar machen 51
– Vergleich mit Replikationssätzen 118
– Vergleich mit Statussätzen 311
– Vorteile 105
– YAML-Definition 105
Replikationsmanager 354
Replikationssätze
– Clusterverbund 607
– Definieren 119
– Eigener Replikationssatz für jede Podinstanz 308
– Einführung 118
– Erstellen 120
– Grundlage von Deployments 287
– Labelselektoren 121
– Mehrere Replikate mit jeweils eigenem Speicher 308
– Statusbehaftete Pods 307
– Vergleich mit Replikationscontrollern 118
– Vergleich mit Statussätzen 311
Replikatsätze
– Controller 363
– Erstellt durch Deployment-Controller 363
– Pods erstellen 363
– Replikatsatzcontroller 355
Repositorys
– Ausgangspunkt für ein Volume 184
– Kubernetes 590
– Privat 187, 247
– Synchronisierung im Sidecar-Container 186
– Überwachen 574
– Volumes synchronisieren 185 f.
– yum 590
requiredDropCapabilities 429 f.
ResourceQuota. Siehe Ressourcenkontingente
Ressourcen
– Aktivitätssonden 101
– Anfordern 442
– Autoskalierung aufgrund der Arbeitsspeichernutzung 486
– Autoskalierung aufgrund der CPU-Nutzung 479
– Benutzerdefiniert 448
– Bessere Ausnutzung 25
– Divisor 252
– Einschränken 14
– Extended Resources 448
– Freigeben 447
– Gesamtressourcen in einem Namespace beschränken 463
– Grafana 472
– Grenzwertbereiche 459
– Grenzwerte 449
– Kapazität von Arbeitsknoten bestimmen 444
– Knotenauswahl 351
– Komprimierbar 449
– Millicore 252
– Mindest- und Höchstwerte 460
– Nutzung auf einzelnen Knoten 469
– Nutzung auf einzelnen Pods 470
– Nutzung überwachen 468
– Nutzung untersuchen 442
– Pods, die auf keinen Knoten passen 445
– Tatsächliche Nutzung 468
– Verbrauch überwachen 26
– Verfügbare Ressourcen auf einem Knoten bestimmen 443
– Verfügbare Ressourcen einschränken 449
– Verlaufsdaten des Verbrauchs 470
– Vom Container wahrgenommene CPU-Kerne 453
– Vom Container wahrgenommener Arbeitsspeicher 453
– Zugewiesene Ressourcen untersuchen 446
– Zuweisbar 445
Ressourcenanforderungen
– Ändern 490
– Arbeitsspeicher 442
– Automatisch einrichten 489
– Benutzerdefinierte Ressourcen 448
– Besten Knoten auswählen 444
– CPU 442
– CPU-Zeitzuteilung 448
– Einfluss auf Scheduler 443
– Gleicher Wert wie Grenzwerte 450
– Podmanifest 442
– QoS-Klassen 454, 456
– Standardanforderungen 458, 460, 462
– Verhältnis zu Grenzwerten 460
Ressourcengrenzwerte
– Festlegen 450
– Grenzwertbereiche 459
– QoS-Klassen 454, 456
– Sicht der Anwendungen 452
– Standardgrenzwerte 458, 460, 462
– Überschreiten 451, 461
– Verhältnis zu Anforderungen 460
– Wiederholt überschreiten 451
– Zu hoch 450
– Zu niedrig 452

Ressourcenkontingente
– Erstellen 463
– Gültigkeitsbereich 467
– Höchstzahl der Objekte im Namespace 466
– Kombination mit Grenzwertbereich 464
– Persistenter Speicher 465
– Plug-in 463
– Podstatus 467
– QoS-Klassen 467
– Untersuchen 464
REST-API
– Einführung 5
– Kommunikation mit dem API-Server 258
– Swagger 272
revisionHistoryLimit 297
Revisionsverlauf 289, 296 f.
RFC 1035 89
rkt
– Containeranzeige 600
– Einführung 18
– Images anzeigen 601
– Kubernetes einrichten 599
– Minikube 600
– OCI-Format 601
– Pods ausführen 600
Rollen
– An Benutzer binden 393
– An Dienstkonto binden 393
– Berechtigungen zum Erstellen und Ändern von Rollen 405
– Clusterrollen 395
– Definition 391
– Erstellen 392
– Namespace 391
– Zugriff 395
Rollenbindungen
– Clusterrollen 396, 400
– Clusterrollenbindungen 397
– Dienstkonten aus anderen Namespaces 394
– Erstellen 393
– Manifest 393
– Zugriff auf Ressourcen in einem Namespace 400
Rollengestützte Zugriffssteuerung. Siehe RBAC
RollingUpdate 290
Rolloutrate 297
root 414, 416, 589
run 30
RunAsAny 427
runAsUser 416, 427

S

Scheduler
– Akzeptable Knoten 351
– Antiaffinitätspräferenzen 516
– Antiaffinitätsregeln 514
– Aufgaben 23
– Benutzerdefinierte Mängel 500
– Besten Knoten auswählen 444
– Bester Knoten für einen Pod 351
– Einfluss von Ressourcenanforderungen 443
– Einführung 22
– Erweiterte Planung 497
– Erweiterte Podplanung 351
– Funktionsweise 350
– Geografische Region 512
– Gründe für fehlgeschlagene Zuteilung ermitteln 446
– Hardwarevoraussetzungen für Zuteilung 503
– Hochverfügbarkeit 375
– Hostports 412
– Kapazität von Arbeitsknoten bestimmen 444
– Knotenaffinität 502
– Knotenprioritäten 505
– LeastRequestedPriority 444
– Mängel 498
– Mehrere Instanzen 375
– Mehrere Scheduler 352
– MostRequestedPriority 444
– Nicht-Produktionspods nicht auf Produktionsknoten ausführen 500
– NoExecute 499
– NoSchedule 499
– Podaffinität 511
– Podaffinitätsregeln 509
– Pods, die auf keinen Knoten passen 445
– Pods zuweisen 363
– PreferNoSchedule 499
– Standardalgorithmus 350
– Tolerierungen 499
– Verfügbare Ressourcen auf einem Knoten bestimmen 443
– Verfügbarkeitszonen 505, 512
– Wartezeit für die Neuzuteilung nach einem Knotenausfall 501
schedulerName 352
SDN 369
Secrects. Siehe Geheimnisse
securityContext 414
Selbstheilung 25, 96
SELinux 415, 590

Sensible Daten
– Geheimnisse 237
– Übergeben 237
sessionAffinity 141
Shell
– Bash 37
– Im Container ausführen 146
– In Container ausführen 37
– MongoDB 192
– Shell-Form/Exec-Form 215
– Shell-Vervollständigung für Aliase 47
Sicherheit
– API-Server 337, 379
– Ausführung als root verhindern 416
– Ausgehenden Datenverkehr einschränken 438
– Authentifizierung 379
– Autorisierung 386
– Benutzer-ID eines Containers 415
– Egress-Regeln 438
– Geheimnisse 237
– HTTPS 164
– Ingress-Regeln 434
– Kernelfähigkeiten 418
– Netzwerk 434
– Nicht-Produktionspods nicht auf Produktionsknoten ausführen 500
– Podsicherheitsrichtlinien 424
– Prinzip der geringstmöglichen Berechtigungen 406
– RBAC 386
– SELinux 415, 590
– Sensible Daten übergeben 237
– Sicherheitskontext von Containern 414
– Speicherung der JSON-Manifeste von Geheimnissen 345
– TLS-Verbindungen 164
– Zertifikate 165
Sicherheitskontext
– Benutzer-ID 415
– Einstellungen 414
– Fähigkeiten 418
– Fähigkeiten entfernen 430
– Fähigkeiten zu allen Containern hinzufügen 430
– fsGroup 422
– Gruppen 422
– Hinzufügbare Fähigkeiten 429
– Podebene 422
– Pods ohne Angabe eines Sicherheitskontexts 415
– Privilegierter Modus 417
– root 416
– runAsUser 416
– Schreiben im Dateisystem verhindern 421
– supplementalGroups 422
Sidecar 186
Sidecar-Container 67
SIGKILL 90, 99
SIGTERM 90, 531, 533
Sitzungsaffinität 141, 156
Skalieren
– Claims löschen 315
Skalierung
– Auslösen 483
– Automatisch 26
– Automatische horizontale Podskalierung 476
– Automatische vertikale Podskalierung 489
– Autoskalierung 476
– Claims an neue Podinstanzen anhängen 316
– Clusterknoten 490
– Deklarativ 116
– Dienstunterbrechung minimieren 493
– Einführung 5
– Ergebnis einsehen 55
– Horizontal 5, 54, 115
– Jobs 130
– Knoten aufgeben 492
– Maximale Skalierungsrate 485
– Mehrschichtige Anwendungen auf mehrere Pods aufteilen 67
– Microservices 6
– Null Replikate 489
– Podausfallvorgabe 493
– Pods ersetzen 284
– Replikationscontroller 115
– Replikationscontroller herunterskalieren 116
– Statussätze 314, 326, 534
– Vertikal 5
– Zusätzliche Knoten anfordern 491
– Zu schnelles Herunterskalieren 314
SNAT 158
Software Defined Network 369
Source-to-Image-Mechanismus 574
spec 71
Speicherklassen
– Auflisten 207
– Claim 205
– Claims ohne Angabe der Speicherklasse 208
– Definieren 204
– Einführung 204
– Leerer String 209
– Namespace 204
– Provisioner 205
– Standardspeicherklasse 207
– storageClassName 209
– Verwenden 206

SRV-Einträge 327
Standardnamespace 87, 89
startingDeadlineSeconds 132
StatefulSet. Siehe Statussätze
Statis
– NotReady 333
status 71
Status
– Anwendungsspezifische Statusmeldungen 543
– Arbeitsknoten 596
– Claims 202
– CrashLoopBackOff 543
– Dienstinstanzen 569
– Knotenstatus 333
– NotReady 596
– OOMKilled 452
– Pending 49, 446
– Persistente Volumes 202
– Pods mit unbekanntem Status 333
– Released 202
– Ressourcenkontingente 467
– Rollout 289
– Running 49
– Steuerebene 341
– Terminating 335
Statussätze
– Aktualisieren 330
– Anwendungen bereitstellen 318
– Claims an neue Podinstanzen anhängen 316
– Claims löschen 315
– Claim-Templates 315
– Einführung 311
– Erstellen 321
– Garantien 316
– Höchstens-eine-Semantik 317
– Individueller Speicher 316
– Knotenausfall 332
– Manifest 320
– Peer-Ermittlung 327
– Persistenter Speicher 314
– Persistente Volumes 319
– PetSets 311
– Pods mit unbekanntem Status 333
– Reguläre Dienste für Haustierpods 326
– Schrittweise Aktualisierung 331
– Schrittweise Erstellung der Pods 321
– Skalieren 314, 326, 534
– Statussatzcontroller 355
– Steuerdienst 320
– Unveränderliche Identität 316
– Vergleich mit Replikationscontrollern 311
– Vergleich mit Replikationssätzen 311
– Verwaiste Claims 534
– Verwenden 317
– volumeClaimTemplates 321
– Zuweisung desselben Speichers zu einer neuen Podinstanz 324
Steuerdienst 312, 320
Steuerebene
– Ausführung als Pods 342
– Bestandteile 22, 340
– Hochverfügbarkeit 374
– Komponenten als Pods ausführen 357
– Koordinatorwahl 376
– Mehrere Instanzen von Komponenten der Steuerebene 342
– Pods auflisten 594
– Status prüfen 341
– Verbundsteuerebene 604
storageClassName 209
stringData 242
subPath 234
supplementalGroups 422, 427
Swagger 272
SYS_ADMIN 429
SYS_MODULE 429
system\
– discovery 398

T

Tabulatorvervollständigung 47
Tags
– Aktualisierung ohne Änderung von Tags 282
– latest 35, 282
– Schichten 35
– Versionsangabe 541
Taints. Siehe Mängel
TCP-Socket-Sonde 97, 167
tenant 436
terminationGracePeriodSeconds 532
terminationMessagePolicy 543
ThirdPartyResource 554
TLS-Zertifikat 164
Tokendatei 381
Tolerierungen
– Anzeigen 499
– Clusterplanung 501
– Gleichheitszeichen 499
– Hinzufügen 500
– Standardtolerierungen 502
– Wartezeit für die Neuzuteilung nach einem Knotenausfall 501
– YAML-Code 500
topologyKey 512

U

Umgebungsvariable
– Konfigurationsdatei 580
– KUBECONFIG 580
Umgebungsvariablen
– Adresse des API-Servers 264
– Alle Einträge einer Konfigurationszuordnung auf einmal übergeben 227
– Auf andere Variablen verweisen 220
– Container 144
– Containerdefinition 220
– Dienste 220
– Dienste finden 144
– Divisor 252
– Downward-API 251
– Einrichten 219
– Geheimniseinträge 246
– Konfigurationseinträge übergeben 226
– Konfigurationsoptionen 214
– Metadaten 251
– MYSQL_ROOT_PASSWORD 214
– Nachteile 221
– Präfix 144, 227
undo 296
USER (Direktive) 415
userspacer-Proxy 358
UTS 13

V

Verbundressourcen 605 f.
Verbundsteuerebene 604
Verfügbarkeitszonen 505, 512
Versionssteuerung
– Entwicklung 549
– Ressourcenmanifeste 549
– Tags 541
Versionsverwaltung
– Anwendungen aktualisieren 276
– Bereitgestellte Version zurücknehmen 294
– Canary-Release 77, 300
– Containerimages 31
– Labels 76
– Rollout fehlerhafter Versionen verhindern 301, 304
– Rollout rückgängig machen 295
– Schrittweise Aktualisierung 278
– Umschalten von alter auf neue Version 278
Veth-Paar 368
Viehherden 311
VirtualBox 585
Virtuelle Maschinen. Siehe VMs
VMs
– Anwendungen ausführen 602
– Betriebssystem installieren 587
– Erstellen 585
– Hypervisor 11
– Klonen 591
– Komponenten isolieren 10
– Master 585
– Nachteile 10
– Namensauflösung 593
– Netzwerkadapter 586
– QCOW2 602
– Startdatenträger 587
– Vergleich mit Containern 11
– Vergleich mit Docker 16
– VM-Images 602
– Vorteile 13
Volume
– Containermetadaten 256
– Lebensdauer 180
volumeClaimTemplates 321
Volumes
– Arten 180
– Atomare Aktualisierung der Dateien 236
– awsElasticBlockStore 193
– Berechtigungen 235
– Claims 195
– configMap 229
– Dateisystem des Knotens 188
– Daten erhalten 522
– downwardAPI 251, 254
– Einführung 178
– Einzelne Konfigurationseinträge verfügbar machen 233
– emptyDir 180 f.
– Fehlende Aktualisierung von Dateien 237
– gcePersistentDisk 191
– GCE Persistent Disk 190
– Gemeinsame Nutzung von Dateiverzeichnissen 65
– Gemeinsam nutzen 422
– gitRepo 184
– hostPath 188
– Inhalt untersuchen 232
– Konfigurationseinträge verwenden 231
– Leeres Verzeichnis 184
– Medium 184
– Mehrere Replikate mit jeweils eigenem Speicher 308
– Mehrere Verzeichnisse in einem gemeinsamen Volume 309
– NFS 194
– Persistente Volumes 195

– Repository als Ausgangspunkt 184
– secret 239
– secret-Volumes im Arbeitsspeicher ablegen 246
– Speichertechnologien 194
– Synchronisieren mit Repository 185 f.
– Synchronisierung im Sidecar-Container 186
– Systempods mit hostPath-Volumes 188
– Volumetypen einschränken 430

W

watch 483
Weave Net 596

Y

YAML
– Aktivitätssonde 97
– API-Dienst 564
– API-Versionsattribut 120
– Autoskalierer 481
– Beendigungsmeldung 542
– Benutzerdefinierte Ressource 555
– Clusterdienstbroker 567
– Clusterdienstklasse 568
– Clusterrolle 396
– CRD-Objekt 555
– Cron-Job 131
– Daemonset 124
– Definition abrufen 73
– Deployment 288
– Deskriptor schreiben 71
– Dienstbindung 570
– Dienste 138
– Dienstinstanz 569
– Endpoints-Objekt 149
– Geheimnis 241
– Headless Service 172
– Hostport 412
– Ingress 161
– Initialisierungscontainer 526
– Instanz einer benutzerdefinierten Ressource 556
– Jobs 128
– Knotenaffinitätsregel 504
– Konfigurationszuordnung 223, 230
– Ksonnet als Alternative 549
– Lebenszyklushook 528, 530
– List-Objekt 319
– Multiportdienst 142
– Namespace erstellen 88
– NodePort-Dienst 152
– Parametrisierbar 573
– Pipe-Zeichen 231
– Podausfallvorgabe 494
– Poddefinition 69
– Pods ohne YAML-Manifest 173
– Post-Start-Hook 528
– Pre-Stop-Hook 530
– Replikationscontroller 105
– Replikationssatz 119
– Rolle 391
– Rollenbindung 393
– Statussatz 320
– Tolerierung 500

Z

Zentrale Protokollierung 544
Zertifikate
– CertificateSigningRequest 165
– Geheimnis 164
– HTTPS 164
– Serverzertifikat prüfen 264
– Signieren 165
– TLS 164
– Zertifizierungsstelle 264
Zugangssteuerungs-Plug-ins
– Aktivieren 425
– Einführung 348
– Grenzwertbereiche 459
– InitialResources 489
– LimitRanger 459
– PodSecurityPolicy 424
– ResourceQuota 463
Zustandsprüfung 25