Burns • Villalba • Strebel • Evenson

Kubernetes
Best Practices

Praktische Anleitungen und Vorlagen zu Grundlagen und fortgeschrittenen Themen

Übersetzung der 2. englischen Auflage

dpunkt.verlag

Brendan Burns ist ein angesehener Engineer bei Microsoft Azure und Mitbegründer des Open-Source-Projekts Kubernetes. Er entwickelt seit mehr als einem Jahrzehnt Cloud-Anwendungen.

Eddie Villalba ist Engineering Manager und Application Platform Practice Lead für Nordamerika bei Google Cloud. Er leitet ein Team von Ingenieuren, das sich darauf konzentriert, Kunden beim Aufbau containeroptimierter Plattformen für skalierbare, zuverlässige verteilte Anwendungen zu unterstützen.

Dave Strebel ist Global Cloud Native Architect bei Microsoft Azure mit Schwerpunkt auf Open Source Cloud und Kubernetes. Er ist stark in das Open-Source-Projekt Kubernetes involviert, unterstützt das Kubernetes-Release-Team und leitet die SIG-Azure.

Lachlan Evenson ist Principal Program Manager im Container Compute Team bei Microsoft Azure. Er hat zahlreichen Menschen beim Einstieg in Kubernetes geholfen, sowohl durch praxisnahe Schulungen als auch mit seinen Vorträgen auf Konferenzen.

Brendan Burns · Eddie Villalba · Dave Strebel · Lachlan Evenson

Kubernetes Best Practices

Praktische Anleitungen und Vorlagen zu Grundlagen und fortgeschrittenen Themen

Übersetzung der 2. englischen Auflage

dpunkt.verlag

Brendan Burns
Eddie Villalba
Dave Strebel
Lachlan Evenson

Übersetzung: Rainer G. Haselier
Lektorat: Sandra Bollenbacher, Alissa Melitzer
Copy-Editing: Petra Heubach-Erdmann, Düsseldorf
Satz: inpunkt[w]o, Wilnsdorf, *www.inpunktwo.de*
Herstellung: Stefanie Weidner
Umschlaggestaltung: Eva Hepper, Silke Braun

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.d-nb.de* abrufbar.

ISBN:
Print 978-3-98889-027-6
PDF 978-3-98890-199-6
ePub 978-3-98890-200-9

1. Auflage 2025

dpunkt.verlag GmbH
Wieblinger Weg 17
69123 Heidelberg

Authorized German translation of the English edition of *Kubernetes Best Practices, 2E*
ISBN 9781098142162 © 2024 Brendan Burns, Eddie Villalba, Dave Strebel, and Lachlan Evenson.
This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

Schreiben Sie uns:
Falls Sie Anregungen, Wünsche und Kommentare haben,
lassen Sie es uns wissen: *hallo@dpunkt.de*.

Die Autoren von *Kubernetes Best Practices* sind Koryphäen im Bereich Cloud Native. Das Buch ist ein Meisterkurs in der Verwaltung von Container-Orchestrierung im großen Maßstab. Mit seinem Themenspektrum von der Einrichtung bis zur Sicherheit ist das Buch eine umfassende Ressource, die nicht nur Wissen vermittelt, sondern auch empowert. Halbieren Sie Ihre Lernkurve und erstellen Sie mit den in dieser Pflichtlektüre vorgestellten bewährten Strategien schneller bessere Anwendungen.

– Joseph Sandoval, Principal Product Manager, Adobe Inc.

Nur weil wir etwas tun können, heißt das nicht, dass wir es auch tun sollten. Cloud Native ist ein umfangreiches Thema und daher gibt es zahlreiche Gelegenheiten, etwas falsch zu machen. Diese Experten fokussieren ihr tiefes Wissen auf die wichtigsten Rezepte, die Ihnen dabei helfen, dass Ihre Kubernetes-Deliveries nicht entgleisen.

– Jonathan Johnson, Cloud Native-Architekt, Vortragsredner, Trainer

Ihr Leitfaden für die Entwicklung erfolgreicher Anwendungen mit Kubernetes, gespickt mit Expertenwissen und Best Practices aus der Praxis.

– Dr. Roland Huß, Senior Principal Software Developer, Red Hat

Eine Fundgrube an Weisheiten über Container-Management von wahren Experten. Zitieren Sie dieses Buch in Meetings! Es geht nicht darum, Ideen zu stehlen – sie wollen, dass Sie dieses Buch lesen.

– Jess Males, DevOps Engineer, TriumphPay

Das Kubernetes-Ökosystem hat sich im Laufe der Zeit erheblich erweitert. Die in diesem exzellenten Leitfaden beschriebenen spezifischen, umsetzbaren Empfehlungen machen die aktuelle Komplexität für die wachsende Community greifbar.

– Bridget Kromhout, Principal Product Manager, Microsoft

Inhalt

Einleitung

Wer dieses Buch lesen sollte

Kubernetes ist der De-facto-Standard für die cloudnative Entwicklung. Es ist ein leistungsstarkes Tool, mit dem sich Ihre nächste Anwendung einfacher entwickeln, schneller bereitstellen und zuverlässiger betreiben lässt. Um die Leistungsfähigkeit von Kubernetes auszuschöpfen, muss es jedoch richtig eingesetzt werden. Dieses Buch richtet sich an alle, die reale Anwendungen auf Kubernetes bereitstellen und daran interessiert sind, Muster und Praktiken zu lernen, die sie für Anwendungen nutzen können, die sich auf Kubernetes aufbauen.

Wichtig ist: Dieses Buch ist keine Einführung in Kubernetes. Wir gehen davon aus, dass Sie mit der Kubernetes-API und den Tools vertraut sind und wissen, wie man einen Kubernetes-Cluster erstellt und mit ihm interagiert. Wenn Sie Kubernetes erlernen möchten, gibt es zahlreiche hervorragende Ressourcen, wie beispielsweise das Buch *Kubernetes – Eine kompakte Einführung* (ebenfalls im dpunkt.verlag erschienen), das genau das ist, was der Titel verspricht, nämlich eine gute und kompakte Einführung in Kubernetes.

Stattdessen ist dieses Buch eine Ressource für alle, die tiefer in die Bereitstellung bestimmter Anwendungen und Workloads auf Kubernetes eintauchen möchten. Das Buch sollte nützlich für Sie sein, gleichgültig, ob Sie gerade dabei sind, Ihre erste Anwendung auf Kubernetes zu implementieren, oder ob Sie Kubernetes bereits seit Jahren verwenden.

Warum wir dieses Buch geschrieben haben

Wir vier haben viel Erfahrung bei der Unterstützung einer Vielzahl von Benutzern, die ihre Anwendungen auf Kubernetes bereitstellen wollen. Wir haben gesehen, wo Menschen Schwierigkeiten haben, und wir haben ihnen geholfen, ihren Weg zum Erfolg zu finden. Beim Schreiben dieses Buches haben wir versucht, diese Erfahrungen festzuhalten, damit noch mehr Menschen Nutzen aus den Lektionen ziehen können, die wir aus diesen realen Erfahrungen gelernt haben. Wir hoffen, dass wir durch die schriftliche Fixierung unserer Erfahrungen unser Wissen skalieren können, damit Sie Ihre Anwendung auf Kubernetes selbstständig erfolgreich bereitstellen und verwalten können.

Was Sie in diesem Buch finden

Obwohl Sie dieses Buch in einer einzigen Sitzung von vorne bis hinten durchlesen könnten, entspricht dies nicht dem, was wir im Hinterkopf hatten. Stattdessen haben wir dieses Buch als Sammlung von einzelnen Kapiteln konzipiert. Jedes Kapitel gibt einen vollständigen Überblick über eine bestimmte Aufgabe, die Sie möglicherweise mit Kubernetes erledigen müssen. Wir gehen davon aus, dass die Leser in das Buch eintauchen, um etwas über ein bestimmtes Thema oder Interesse zu erfahren, dann das Buch in Ruhe lassen, um erst zurückzukehren, wenn ein neues Thema mit neuen Fragen auftaucht.

Trotz dieses eigenständigen Ansatzes ziehen sich einige Themen durch das gesamte Buch. Es gibt mehrere Kapitel über die Entwicklung von Anwendungen auf Kubernetes. Kapitel 2 behandelt Entwickler-Workflows. Kapitel 5 befasst sich mit kontinuierlicher Integration und Tests. Kapitel 15 befasst sich mit dem Aufbau übergeordneter Plattformen auf Kubernetes, und Kapitel 16 behandelt die Verwaltung von Zuständen und zustandsbehafteten Anwendungen. Neben der Entwicklung von Anwendungen gibt es auch mehrere Kapitel zum Betrieb von Services in Kubernetes. Kapitel 1 befasst sich mit der Einrichtung eines grundlegenden Service, und Kapitel 3 behandelt die Überwachung und Metriken. Kapitel 4 befasst sich mit der Konfigurationsverwaltung und Kapitel 6 mit der Versionierung und den Releases. Kapitel 7 befasst sich mit der globalen Bereitstellung Ihrer Anwendung.

Es gibt auch mehrere Kapitel über die Clusterverwaltung, darunter Kapitel 8 über die Ressourcenverwaltung, Kapitel 9 über die Vernetzung, Kapitel 10 über die Pod-Sicherheit, Kapitel 11 über Policy und Governance, Kapitel 12 über die Verwaltung mehrerer Cluster und Kapitel 17 über die Zugangskontrolle und Autorisierung. Schließlich sind einige Kapitel wirklich unabhängig; diese behandeln maschinelles Lernen (Kap. 14) und die Integration mit externen Services (Kap. 13).

Obwohl es nützlich sein kann, alle Kapitel zu lesen, bevor Sie das Thema in der Praxis anwenden, hoffen wir, dass Sie dieses Buch als Nachschlagewerk betrachten. Es soll Ihnen als Leitfaden dienen, wenn Sie diese Themen in der Praxis umsetzen.

Neu in dieser Ausgabe

Wir haben die erste Auflage dieses Buches um vier neue Kapitel erweitert, die sich mit neuen Tools und Mustern befassen, während sich Kubernetes weiterentwickelt. Diese neuen Kapitel sind Kapitel 18 über GitOps, Kapitel 19 über Sicherheit, Kapitel 20 über Chaosexperimente und andere Testverfahren sowie Kapitel 21, in dem es um die Implementierung eines Operators geht.

In diesem Buch verwendete Konventionen

Die folgenden typografischen Konventionen werden in diesem Buch genutzt:

- *Kursiv*
  Für neue Begriffe, URLs, E-Mail-Adressen, Dateinamen und Dateierweiterungen.
- `Nichtproportionalschrift`
  Für Programmlistings, aber auch für Codefragmente in Absätzen, wie etwa Variablen- oder Funktionsnamen, Datenbanken, Datentypen, Umgebungsvariablen, Anweisungen und Schlüsselwörter.
- **`fette Nichtproportionalschrift`**
  Für Befehle und anderen Text, der genau so vom Benutzer eingegeben werden sollte.
- *`kursive Nichtproportionalschrift`*
  Für Text, der vom Benutzer durch eigene Werte ersetzt werden sollte.

Tipp

Dieses Element steht für einen Tipp oder eine Anregung.

Hinweis

Dieses Element kennzeichnet einen allgemeinen Hinweis.

Achtung

Dieses Element kennzeichnet eine Warnung oder einen Warnhinweis.

Codebeispiele verwenden

Zusätzliches Material (Codebeispiele, Übungen usw.) steht zum Herunterladen unter *https://oreil.ly/KBPsample* bereit.

Dieses Buch ist dazu da, Ihnen beim Erledigen Ihrer Arbeit zu helfen. Im Allgemeinen dürfen Sie die Codebeispiele aus diesem Buch in Ihren eigenen Programmen und der dazugehörigen Dokumentation verwenden. Sie müssen uns dazu nicht um Erlaubnis fragen, solange Sie nicht einen beträchtlichen Teil des Codes reproduzieren. Beispielsweise benötigen Sie keine Erlaubnis, um ein Programm zu schreiben, in dem mehrere Codefragmente aus diesem Buch vorkommen. Wollen Sie dagegen einen Datenträger mit Beispielen aus Büchern von der dpunkt.verlag GmbH verkaufen oder

verteilen, benötigen Sie eine Erlaubnis. Eine Frage zu beantworten, indem Sie aus diesem Buch zitieren und ein Codebeispiel wiedergeben, benötigt keine Erlaubnis. Eine beträchtliche Menge Beispielcode aus diesem Buch in die Dokumentation Ihres Produkts aufzunehmen, bedarf hingegen einer Erlaubnis.

Wir freuen uns über Zitate, verlangen diese aber nicht. Ein Zitat enthält Titel, Autor, Verlag und ISBN. Beispiel: »*Kubernetes Best Practices* von Brendan Burns, Eddie Villalba, Dave Strebel und Lachlan Evenson. Copyright 2024 dpunkt.verlag GmbH, 978-3-98889-027-6.«

Wenn Sie glauben, dass Ihre Verwendung von Codebeispielen über die übliche Nutzung hinausgeht oder außerhalb der oben beschriebenen Nutzungsbedingungen liegt, kontaktieren Sie uns bitte unter *hallo@dpunkt.de*.

Wie Sie uns erreichen

Mit Anmerkungen, Fragen oder Verbesserungsvorschlägen zu diesem Buch können Sie sich jederzeit an den Verlag wenden:

hallo@dpunkt.de

Bitte beachten Sie, dass über unsere E-Mail-Adresse kein Software-Support angeboten wird.

Danksagungen

Brendan möchte sich bei seiner wunderbaren Familie, Robin, Julia und Ethan, für die Liebe und Unterstützung bei allem, was er tut, bedanken; bei der Kubernetes-Community, ohne die nichts von alledem möglich wäre; und bei seinen fabelhaften Co-Autoren, ohne die dieses Buch nicht existieren würde.

Dave möchte sich bei seiner wunderschönen Frau Jen und seinen drei Kindern Max, Maddie und Mason für ihre Unterstützung bedanken. Er möchte sich auch bei der Kubernetes-Community für all die Ratschläge und die Hilfe bedanken, die sie im Laufe der Jahre geleistet haben. Schließlich möchte er seinen Mitautoren dafür danken, dass sie dieses Abenteuer in die Tat umgesetzt haben.

Lachlan möchte sich bei seiner Frau und seinen drei Kindern für ihre Liebe und Unterstützung bedanken. Er möchte sich auch bei allen Mitgliedern der Kubernetes-Community bedanken, einschließlich der wunderbaren Menschen, die sich die Zeit genommen haben, ihn im Laufe der Jahre zu unterrichten. Ein besonderer Dank geht an Joseph Sandoval für seine Mentorenschaft. Und schließlich möchte er seinen fantastischen Mitautoren dafür danken, dass sie dieses Buch möglich gemacht haben.

Eddie möchte sich bei seiner Frau Sandra für ihre unermüdliche Unterstützung, Liebe und Ermutigung während des Schreibprozesses bedanken. Er möchte auch seiner Tochter Giavanna dafür danken, dass sie ihn motiviert hat, ein Vermächtnis zu hinterlassen, damit sie stolz auf ihren Vater sein kann. Schließlich möchte er sich bei

der Kubernetes-Community und seinen Co-Autoren bedanken, die ihm auf seinem Weg, cloudnativ zu werden, immer ein Wegweiser waren.

Wir alle möchten Virginia Wilson für ihre Arbeit bei der Entwicklung des Manuskripts und ihre Hilfe bei der Zusammenführung all unserer Ideen danken und Jill Leonard für ihre Anleitung bei der 2. Ausgabe. Schließlich möchten wir Bridget Kromhout, Bilgin Ibryam, Roland Huß, Justin Domingus, Jess Males und Jonathan Johnson für ihre Aufmerksamkeit beim Feinschliff danken.

1 Einen einfachen Service einrichten

Dieses Kapitel beschreibt das Verfahren zum Einrichten einer einfachen mehrschichtigen Anwendung in Kubernetes. Das Beispiel, das wir durchgehen werden, besteht aus zwei Schichten: einer einfachen Webanwendung und einer Datenbank. Auch wenn es sich hierbei nicht um die komplizierteste Anwendung handelt, ist es ein guter Ausgangspunkt, um die Verwaltung einer Anwendung in Kubernetes zu erlernen.

1.1 Die Anwendung im Überblick

Die Anwendung für unser Beispiel ist recht simpel. Es handelt sich um einen einfachen Journaldienst, der folgende Details aufweist:

- Die Anwendung hat einen separaten statischen Dateiserver, der NGINX verwendet.
- Sie verfügt über ein RESTful Application Programming Interface (API) *https://some- host-name.io/api* im Pfad */api*.
- Sie besitzt auf der Haupt-URL *https://some-host-name.io* einen Dateiserver.
- Sie verwendet den Dienst Let's Encrypt (*https://letsencrypt.org*) für die Verwaltung von Secure Sockets Layer (SSL).

Abbildung 1–1 zeigt ein Diagramm dieser Anwendung. Machen Sie sich keine Sorgen, wenn Sie nicht auf Anhieb alle einzelnen Teile verstehen; sie werden im Laufe des Kapitels detaillierter erklärt. Wir werden den Aufbau dieser Anwendung Schritt für Schritt durchgehen, und verwenden zunächst YAML-Konfigurationsdateien und dann Helm-Charts.

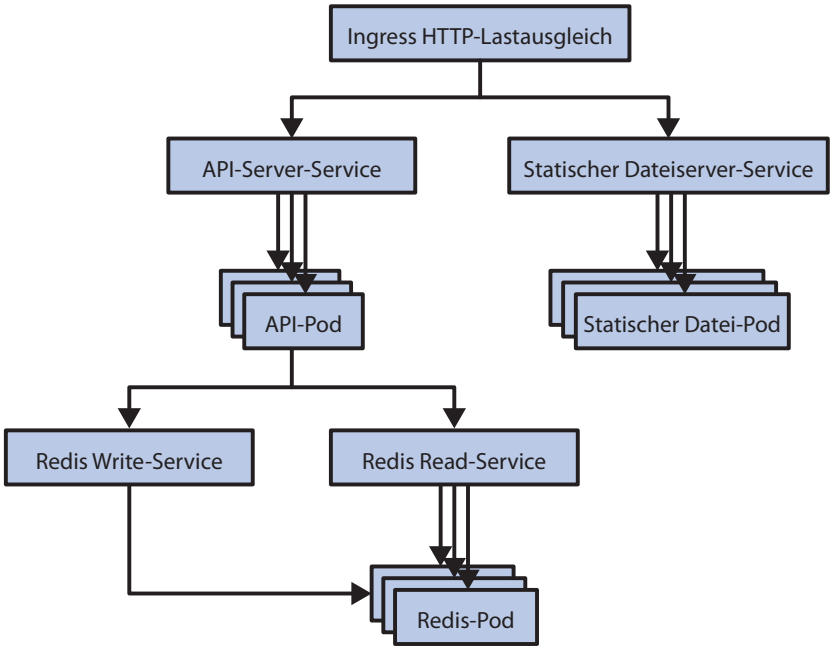

Abb. 1–1 *Ein Diagramm unseres Journaldienstes, wie er in Kubernetes implementiert ist*

1.2 Konfigurationsdateien verwalten

Bevor wir uns im Detail mit dem Aufbau dieser Anwendung in Kubernetes befassen, sollten wir beschreiben, wie wir die Konfigurationen selbst verwalten. Bei Kubernetes wird alles *deklarativ* repräsentiert. Das bedeutet, dass Sie den gewünschten Zustand der Anwendung im Cluster aufschreiben (im Allgemeinen in YAML- oder JSON-Dateien). Diese deklarierten gewünschten Zustände definieren alle Teile Ihrer Anwendung. Dieser deklarative Ansatz ist einem *imperativen* Ansatz vorzuziehen, bei dem der Zustand Ihres Clusters die Summe einer Reihe von Änderungen am Cluster ist. Wenn ein Cluster imperativ konfiguriert ist, ist es schwierig zu verstehen und zu reproduzieren, wie er in diesen Zustand gekommen ist. Hierdurch ist es dann auch schwierig, Probleme mit Ihrer Anwendung zu verstehen oder zu beheben.

Bei der Deklaration des Zustands Ihrer Anwendung wird in der Regel YAML gegenüber JSON bevorzugt, obwohl Kubernetes beide Formate unterstützt. Der Grund dafür ist, dass YAML etwas weniger ausführlich ist und von Menschen besser bearbeitet werden kann als JSON. Es ist jedoch zu beachten, dass YAML auf Einrückungen reagiert; häufig lassen sich Fehler in Kubernetes-Konfigurationen auf eine falsche Einrückung in YAML zurückführen. Wenn sich die Dinge nicht wie erwartet verhalten, ist die Überprüfung der Einrückungen ein guter Ansatzpunkt für die Fehlersuche. Die meisten Editoren unterstützen die Syntaxhervorhebung sowohl für JSON als auch für YAML. Wenn Sie mit diesen Dateien arbeiten, ist es eine gute Idee, solche

Tools zu installieren, um in Ihren Konfigurationen sowohl Autoren- als auch Dateifehler leichter zu finden. Es gibt auch eine hervorragende Erweiterung für Visual Studio Code, die eine umfassendere Fehlerprüfung für Kubernetes-Dateien unterstützt.

Da der deklarative Zustand, der in diesen YAML-Dateien enthalten ist, als Source of Truth für Ihre Anwendung dient, ist die korrekte Verwaltung dieses Zustands entscheidend für den Erfolg Ihrer Anwendung. Wenn Sie den gewünschten Zustand Ihrer Anwendung ändern, müssen Sie in der Lage sein, die Änderungen zu verwalten, zu überprüfen, ob sie korrekt sind, zu kontrollieren, wer die Änderungen vorgenommen hat, und im Falle eines Fehlers die Änderungen zurückzunehmen. Glücklicherweise haben wir im Rahmen der Softwareentwicklung bereits die notwendigen Werkzeuge entwickelt, um sowohl Änderungen am deklarativen Zustand als auch die Prüfung und das Rollback zu verwalten. Die bewährten Verfahren zur Versionsverwaltung und für Code-Reviews lassen sich direkt auf die Verwaltung des deklarativen Zustands Ihrer Anwendung anwenden.

Heutzutage speichern die meisten Leute ihre Kubernetes-Konfigurationen in Git. Auch wenn die spezifischen Details des Versionskontrollsystems unwichtig sind, erwarten viele Tools im Kubernetes-Ökosystem Dateien in einem Git-Repository. Beim Code-Review gibt es eine viel größere Heterogenität; obwohl GitHub offensichtlich sehr beliebt ist, verwenden andere On-Premises-Tools oder Dienste für das Code-Review. Unabhängig davon, wie Sie das Code-Review für die Konfiguration Ihrer Anwendung implementieren, sollten Sie sie mit der gleichen Sorgfalt und Konzentration behandeln, die Sie für die Verwaltung des Quellcodes anwenden.

Wenn es darum geht, das Dateisystem für Ihre Anwendung einzurichten, lohnt es sich, die mit dem Dateisystem gelieferte Verzeichnisstruktur zu verwenden, um Ihre Komponenten zu organisieren. In der Regel wird ein einzelnes Verzeichnis genutzt, um einen Anwendungsdienst zusammenzufassen. Die Definition dessen, was einen Anwendungsdienst ausmacht, kann von Team zu Team unterschiedlich sein, aber im Allgemeinen handelt es sich um einen Service, der von einem Team aus 8 bis 12 Personen entwickelt wird. Innerhalb dieses Verzeichnisses werden Unterverzeichnisse für Unterkomponenten der Anwendung eingesetzt.

Für unsere Anwendung legen wir die Verzeichnisstruktur für die Dateien wie folgt an:

```
journal/
  frontend/
  redis/
  fileserver/
```

In jedem Verzeichnis befinden sich die konkreten YAML-Dateien, die zur Definition des Service benötigt werden. Wie Sie später sehen werden, wird dieses Dateilayout komplizierter, wenn wir beginnen, unsere Anwendung in mehreren verschiedenen Regionen oder Clustern einzusetzen.

1.3 Mit Deployments einen replizierten Service erstellen

Um unsere Anwendung zu beschreiben, beginnen wir mit dem Frontend und arbeiten uns nach unten vor. Die Frontend-Anwendung für das Journal ist eine in TypeScript implementierte Node.js-Anwendung. Die komplette Anwendung ist zu umfangreich, um sie in das Buch aufzunehmen, daher haben wir sie auf unseren GitHub hochgeladen (*https://github.com/brendandburns/kbp-sample*). Dort finden Sie auch den Code für zukünftige Beispiele, es lohnt sich also, für diese URL ein Lesezeichen zu setzen. Die Anwendung stellt einen HTTP-Dienst auf Port 8080 bereit, der Anfragen an den Pfad */api/** weiterleitet und das Redis-Backend verwendet, um die aktuellen Journaleinträge hinzuzufügen, zu löschen oder zurückzugeben. Wenn Sie die folgenden YAML-Beispiele auf Ihrem lokalen Rechner durcharbeiten möchten, sollten Sie diese Anwendung mithilfe des Dockerfiles in ein Container-Image erstellen und in Ihr eigenes Image-Repository pushen. Anstatt den Namen unserer Beispieldatei zu nehmen, sollten Sie dann in Ihren Code den Namen Ihres Container-Image aufnehmen.

1.3.1 Best Practices für die Verwaltung von Images

Obwohl die Erstellung und Pflege von Container-Images im Allgemeinen den Rahmen dieses Buches sprengen würde, lohnt es sich, einige allgemeine Best Practices für die Erstellung und Benennung von Images aufzuzeigen. Im Allgemeinen kann der Image-Erstellungsprozess für »Supply-Chain-Angriffe« anfällig sein. Bei solchen Angriffen injiziert ein böswilliger Benutzer Code oder Binärdateien in eine Abhängigkeit von einer vertrauenswürdigen Quelle, die dann in Ihre Anwendung eingebaut wird. Aufgrund des Risikos solcher Angriffe ist es wichtig, dass Sie bei der Erstellung Ihrer Images nur auf bekannte und vertrauenswürdige Image-Anbieter zurückgreifen. Alternativ können Sie auch alle Ihre Images von Grund auf neu erstellen. Bei einigen Sprachen (z. B. Go), die statische Binärdateien erstellen können, ist die Erstellung von Grund auf einfach, bei interpretierten Sprachen wie Python, JavaScript oder Ruby ist dies jedoch wesentlich komplizierter.

Die anderen Best Practices für Images beziehen sich auf die Namensgebung. Obwohl die Version eines Container-Images in einer Image-Registry theoretisch veränderbar ist, sollten Sie das Versions-Tag als unveränderbar behandeln. Insbesondere ist eine Kombination aus der semantischen Version und dem SHA-Hash des Commits, in dem das Image erstellt wurde, eine gute Praxis für die Benennung von Images (z. B. *v1.0.1-bfeda01f*). Wenn Sie keine Image-Version angeben, wird standardmäßig `latest` verwendet. Obwohl dies in der Entwicklung praktisch sein kann, ist es eine schlechte Idee für den produktiven Einsatz, da `latest` jedes Mal, wenn ein neues Image erstellt wird, verändert wird.

1.3.2 Replizierte Anwendung erstellen

Unsere Frontend-Anwendung ist *zustandslos*; sie verlässt sich in Bezug auf ihren Zustand vollständig auf das Redis-Backend. Daher können wir sie beliebig replizieren, ohne den Datenverkehr zu beeinträchtigen. Obwohl es unwahrscheinlich ist, dass unsere Anwendung in großem Umfang genutzt wird, ist es dennoch eine gute Idee, mit mindestens zwei Replikaten zu arbeiten, damit Sie einen unerwarteten Absturz bewältigen oder eine neue Version der Anwendung ohne Ausfallzeiten ausrollen können.

In Kubernetes ist die ReplicaSet-Ressource diejenige, die direkt die Replikation einer bestimmten Version Ihrer containerisierten Anwendung verwaltet. Da sich die Version aller Anwendungen im Laufe der Zeit ändert, wenn Sie den Code ändern, ist es keine optimale Methode, direkt ein ReplicaSet zu verwenden. Stattdessen nutzen Sie die Ressource *Deployment*. Ein Deployment kombiniert die Replikationsfunktionen von ReplicaSet mit der Versionierung und der Möglichkeit, ein gestuftes Rollout durchzuführen. Durch ein Deployment können Sie die in Kubernetes integrierten Werkzeuge nutzen, um von einer Version der Anwendung zur nächsten zu wechseln.

Die Kubernetes-Deployment-Ressource für unsere Anwendung sieht wie folgt aus:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    # Alle Pods im Deployment verwenden dieses Label
    app: frontend
  name: frontend
  namespace: default
spec:
  # Wir sollten aus Gründen der Zuverlässigkeit immer mindestens zwei Replikate
einsetzen
  replicas: 2
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - image: my-repo/journal-server:v1-abcde
        imagePullPolicy: IfNotPresent
        name: frontend
        # TODO: Ermitteln, wie hoch der tatsächliche Ressourcenbedarf ist
        resources:
          request:
            cpu: "1.0"
            memory: "1G"
          limits:
            cpu: "1.0"
            memory: "1G"
```

Bei diesem Deployment sind mehrere Dinge zu beachten. Erstens verwenden wir Labels, um das Deployment sowie die ReplicaSets und die Pods zu identifizieren, die das Deployment erstellt. Wir haben das Label `app: frontend` zu all diesen Ressourcen hinzugefügt, damit wir alle Ressourcen für eine bestimmte Schicht in einer einzigen Anfrage untersuchen können. Sie werden sehen, dass wir beim Hinzufügen weiterer Ressourcen genauso vorgehen werden.

Außerdem haben wir an einigen Stellen in das YAML Kommentare eingefügt. Obwohl diese Kommentare nicht in die auf dem Server gespeicherte Kubernetes-Ressource einfließen, dienen sie, genau wie Kommentare im Code, als Orientierungshilfe für Personen, die sich diese Konfiguration zum ersten Mal ansehen.

Sie sollten weiterhin beachten, dass wir für die Container im Deployment für die Ressourcenanforderungen sowohl Request als auch Limit angegeben und für Request und Limit die gleichen Werte verwendet haben. Wenn eine Anwendung ausgeführt wird, ist Request (Anforderung) die Reservierung, die auf dem Host-Rechner, auf dem sie läuft, garantiert wird. Limit (Obergrenze) ist die maximale Ressourcennutzung, die dem Container zugestanden wird. Wenn Sie am Anfang Request gleich Limit setzen, führt dies zu einem möglichst vorhersehbaren Verhalten Ihrer Anwendung. Diese Vorhersagbarkeit geht jedoch zulasten der Ressourcennutzung. Da die Einstellung Request gleich Limit Ihre Anwendungen daran hindert, zu viel Zeit einzuplanen oder zu viele ungenutzte Ressourcen zu verbrauchen, werden Sie nicht in der Lage sein, eine maximale Auslastung zu erreichen, es sei denn, Sie stimmen Request und Limit sehr, sehr sorgfältig aufeinander ab. Wenn Sie das Kubernetes-Ressourcenmodell besser verstehen, können Sie Request und Limit für Ihre Anwendung unabhängig voneinander anpassen. Die meisten Benutzer vertreten jedoch die Auffassung, dass eine stabile Vorhersagbarkeit die geringere Ressourcenausnutzung wert ist.

Wie unser Kommentar andeutet, ist es oft schwierig, die richtigen Werte für diese Ressourcenobergrenzen zu ermitteln. Ein guter Ansatz ist, die Schätzungen zunächst zu hoch anzusetzen und sie dann mithilfe des Monitorings auf die richtigen Werte abzustimmen. Wenn Sie jedoch einen neuen Service einführen, sollten Sie bedenken, dass Ihr Ressourcenbedarf beim ersten großen Datenverkehr wahrscheinlich erheblich ansteigen wird. Darüber hinaus gibt es einige Sprachen, insbesondere Sprachen mit Garbage Collection, die munter den gesamten verfügbaren Speicher verbrauchen, was es schwierig machen kann, das richtige Minimum für den Speicher zu bestimmen. In diesem Fall kann eine Form der binären Suche notwendig sein, aber denken Sie daran, dies in einer Testumgebung zu tun, damit es sich nicht auf Ihre Produktivumgebung auswirkt!

Nachdem wir nun die Deployment-Ressource definiert haben, checken wir sie in die Versionskontrolle ein und stellen sie in Kubernetes bereit:

```
git add frontend/deployment.yaml
git commit -m "Deployment hinzugefügt" frontend/deployment.yaml
kubectl apply -f frontend/deployment.yaml
```

Es ist ebenfalls eine Best Practice, sicherzustellen, dass der Inhalt Ihres Clusters genau mit dem Inhalt Ihrer Quellcodeverwaltung übereinstimmt. Das beste Muster dafür ist die Verwendung eines GitOps-Ansatzes und die Bereitstellung für die Produktion nur von einem bestimmten Zweig Ihrer Versionsverwaltung aus vorzunehmen. Hierzu nutzen Sie die Automatisierung der kontinuierlichen Integration/kontinuierlichen Bereitstellung (CI/CD). So ist gewährleistet, dass Quellcodeverwaltung und Produktion übereinstimmen. Obwohl eine vollständige CI/CD-Pipeline für eine einfache Anwendung übertrieben erscheinen mag, ist die Automatisierung an sich, unabhängig von der Zuverlässigkeit, die sie bietet, in der Regel den Zeitaufwand für ihre Einrichtung wert. Außerdem lässt sich CI/CD im Nachhinein nur sehr schwer in eine bestehende, imperativ bereitgestellte Anwendung einbauen.

Auf diese mit YAML erstellte Anwendungsbeschreibung werden wir in späteren Abschnitten zurückkommen, um weitere Elemente wie die ConfigMap und Secrets-Volumes sowie die Pod Quality of Service zu untersuchen.

1.4 Externen Ingress für HTTP-Verkehr einrichten

Die Container für unsere Anwendung sind zwar jetzt bereitgestellt, aber es ist derzeit noch nicht möglich, dass jemand auf die Anwendung zugreift. Standardmäßig sind die Cluster-Ressourcen nur innerhalb des Clusters selbst verfügbar. Um unsere Anwendung der Außenwelt zugänglich zu machen, müssen wir einen Service und einen Load Balancer erstellen, um eine externe IP-Adresse bereitzustellen und den Datenverkehr zu unseren Containern zu leiten. Für die externe Bereitstellung werden wir zwei Kubernetes-Ressourcen verwenden.

Die erste Ressource ist ein Service, der für einen Lastausgleich des Datenverkehrs des Transmission Control Protocol (TCP) oder des User Datagram Protocol (UDP) sorgt. In unserem Fall nutzen wir das TCP-Protokoll. Die zweite ist eine Ingress-Ressource, die einen HTTP(S)-Lastausgleich mit intelligentem Routing von Anfragen auf der Grundlage von HTTP-Pfaden und Hosts bietet. Bei einer einfachen Anwendung wie dieser werden Sie sich vielleicht fragen, warum wir die komplexere Ingress-Ressource verwenden, aber wie Sie in späteren Abschnitten sehen werden, werden selbst bei dieser einfachen Anwendung HTTP-Anfragen von zwei verschiedenen Diensten bedient. Darüber hinaus bietet ein Ingress an der Schnittstelle nach außen die Flexibilität, die für zukünftige Erweiterungen unseres Dienstes erforderlich ist.

Hinweis

Die Ingress-Ressource ist eine der älteren Ressourcen in Kubernetes, und im Laufe der Jahre wurden zahlreiche Probleme mit der Art und Weise, wie sie den HTTP-Zugriff auf Microservices modelliert, gelöst. Dies hat zur Entwicklung der Gateway-API für Kubernetes geführt. Die Gateway-API wurde als Erweiterung für Kubernetes entwickelt und erfordert in Ihrem Cluster die Installation zusätzlicher Komponenten. Wenn Sie feststellen, dass Ingress Ihre Anforderungen nicht erfüllt, sollten Sie in Erwägung ziehen, zur Gateway-API zu wechseln.

Bevor die Ingress-Ressource definiert werden kann, muss ein Kubernetes-Service vorhanden sein, auf den der Ingress verweisen kann. Wir werden Labels verwenden, um den Dienst zu den Pods zu leiten, die wir im vorherigen Abschnitt erstellt haben. Der Service ist wesentlich einfacher zu definieren als das Deployment und sieht wie folgt aus:

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: frontend
  name: frontend
  namespace: default
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 8080
  selector:
    app: frontend
  type: ClusterIP
```

Nachdem Sie den Service definiert haben, können Sie eine Ingress-Ressource definieren. Im Gegensatz zu Service-Ressourcen muss für Ingress im Cluster ein Ingress-Controller-Container ausgeführt werden. Sie haben die Wahl zwischen verschiedenen Implementierungen, die entweder von Ihrem Cloud-Provider angeboten oder mit Open-Source-Servern implementiert werden. Wenn Sie sich für die Installation eines Open-Source-Ingress-Anbieters entscheiden, ist es eine gute Idee, den Helm-Paketmanager (*https://helm.sh*) zu verwenden, um ihn zu installieren und zu warten. Die Ingress-Provider `nginx` oder `haproxy` sind beliebte Optionen:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend-ingress
spec:
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 8080
```

Nachdem unsere Ingress-Ressource erstellt wurde, ist unsere Anwendung bereit, den Datenverkehr von Webbrowsern aus der ganzen Welt zu bedienen. Als Nächstes werden wir uns ansehen, wie Sie Ihre Anwendung für eine einfache Konfiguration und Anpassung einrichten können.