



Red Hat OpenShift Development II: Containerizing Applications



OCP 4.6 DO288

Red Hat OpenShift Development II: Containerizing Applications

Ausgabe 2 20210818

Veröffentlicht 20210818

Autoren: Zach Guterman, Richard Allred, Ricardo Jun,
Ravishankar Srinivasan, Fernando Lozano, Ivan Chavero,
Dan Kolepp, Jordi Sola Alaball, Manuel Aude Morales,
Eduardo Ramírez Martínez, Guy Bianco, Randy Thomas,
Marek Czernek

Editor: Sam Ffrench, David O'Brien, Seth Kenlon

Copyright © 2019 Red Hat, Inc.

The contents of this course and all its modules and related materials, including handouts to audience members, are
Copyright © 2019 Red Hat, Inc.

No part of this publication may be stored in a retrieval system, transmitted or reproduced in any way, including, but not limited to, photocopy, photograph, magnetic, electronic or other record, without the prior written permission of Red Hat, Inc.

This instructional program, including all material provided herein, is supplied without any guarantees from Red Hat, Inc. Red Hat, Inc. assumes no liability for damages or legal action arising from the use or misuse of contents or details contained herein.

If you believe Red Hat training materials are being used, copied, or otherwise improperly distributed, please send email to training@redhat.com or phone toll-free (USA) +1 (866) 626-2994 or +1 (919) 754-3700.

Red Hat, Red Hat Enterprise Linux, the Red Hat logo, JBoss, OpenShift, Fedora, Hibernate, Ansible, CloudForms, RHCA, RHCE, RHCSA, Ceph, and Gluster are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

XFS® is a registered trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack® Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation or the OpenStack community.

All other trademarks are the property of their respective owners.

Mitwirkende: Grega Bremec, Sajith Sugathan, Dave Sacco, Rob Locke, Bowe Strickland, Rudolf Kastl, Chris Tusa, Joel Birchler, Chetan Tiwary

Dokumentkonventionen	vii
Einführung	ix
Red Hat OpenShift Development II: Containerizing Applications	ix
Informationen zur Kursumgebung	x
Landessprachliche Anpassung	xiv
1. Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster	1
Einführung in OpenShift Container Platform 4	2
Quiz: Einführung in OpenShift 4	8
Angeleitete Übung: Konfigurieren der Kursumgebung	12
Bereitstellen einer Anwendung auf einem OpenShift-Cluster	22
Angeleitete Übung: Bereitstellen einer Anwendung auf einem OpenShift-Cluster	32
Verwalten von Anwendungen mit der Web Console	40
Angeleitete Übung: Verwalten einer Anwendung mit der Web Console	45
Verwalten von Anwendungen mit der CLI	53
Angeleitete Übung: Verwalten einer Anwendung mit der CLI	59
Praktische Übung: Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster	69
Zusammenfassung	78
2. Entwerfen containerisierter Anwendungen für OpenShift	79
Auswählen eines Ansatzes für die Containerisierung	80
Quiz: Auswählen eines Ansatzes für die Containerisierung	84
Erstellen von Container-Images mit erweiterten Containerfile-Anweisungen	92
Angeleitete Übung: Erstellen von Container-Images mit erweiterten Containerfile-Anweisungen	100
Einfügen von Konfigurationsdaten in eine Anwendung	109
Angeleitete Übung: Einfügen von Konfigurationsdaten in eine Anwendung	117
Praktische Übung: Entwerfen containerisierter Anwendungen für OpenShift	125
Zusammenfassung	136
3. Veröffentlichen von Enterprise Container-Images	137
Verwalten von Images in einer Unternehmens-Registry	138
Angeleitete Übung: Verwenden einer Unternehmens-Registry	147
Zulassen des Zugriffs auf die OpenShift-Registry	153
Angeleitete Übung: Zulassen des Zugriffs auf die OpenShift-Registry	157
Erstellen von Image-Streams	161
Angeleitete Übung: Erstellen eines Image-Streams	169
Praktische Übung: Veröffentlichen von Enterprise Container-Images	173
Zusammenfassung	182
4. Verwalten von Builds auf OpenShift	183
Beschreiben des Red Hat OpenShift-Build-Prozesses	184
Quiz: Der OpenShift-Build-Prozess	188
Verwalten von Anwendungs-Builds	194
Angeleitete Übung: Verwalten von Anwendungs-Builds	198
Auslösen von Builds	205
Angeleitete Übung: Auslösen von Builds	208
Implementieren des Build-Hook „post-commit“	214
Angeleitete Übung: Implementieren des Build-Hook „post-commit“	216
Praktische Übung: Erstellen von Anwendungen für OpenShift	221
Zusammenfassung	230
5. Anpassen von Source-to-Image-Builds	231
Beschreiben der Source-to-Image-Architektur	232
Quiz: Beschreiben der Source-to-Image-Architektur	238
Anpassen eines vorhandenen S2I-Builder-Images	242

Angeleitete Übung: Anpassen von S2I-Builds	246
Erstellen eines S2I-Basis-Images	252
Angeleitete Übung: Erstellen eines S2I-Basis-Images	258
Praktische Übung: Anpassen von Source-to-Image-Builds	269
Zusammenfassung	283
6. Bereitstellen von Anwendungen mit mehreren Containern	285
Erläutern von OpenShift-Vorlagen	286
Quiz: Erläutern von OpenShift-Vorlagen	291
Erstellen eines Helm-Chart	293
Angeleitete Übung: Erstellen eines Helm-Chart	297
Anpassen einer Bereitstellung mit Kustomize	303
Angeleitete Übung: Anpassen von Bereitstellungen mit Kustomize	307
Praktische Übung: Bereitstellen von Anwendungen mit mehreren Containern	314
Zusammenfassung	327
7. Verwalten der Anwendungsbereitstellungen	329
Überwachen des Anwendungszustands	330
Angeleitete Übung: Aktivieren von Probes	336
Auswählen der geeigneten Deploymentstrategie	344
Angeleitete Übung: Implementieren einer Deploymentstrategie	349
Verwalten von Anwendungsbereitstellungen mit CLI-Befehlen	357
Angeleitete Übung: Verwalten der Anwendungsbereitstellungen	363
Praktische Übung: Verwalten der Anwendungsbereitstellungen	370
Zusammenfassung	381
8. Erstellen von Anwendungen für OpenShift	383
Integrieren externer Services	384
Angeleitete Übung: Integrieren eines externen Services	387
Containerisieren von Services	391
Angeleitete Übung: Containerisieren von Nexus als ein Service	399
Bereitstellen Cloud-nativer Anwendungen mit JKube	409
Angeleitete Übung: Bereitstellen einer Anwendung mit JKube	415
Praktische Übung: Erstellen von Cloud-nativen Anwendungen für OpenShift	426
Zusammenfassung	436
9. Ausführliche Wiederholung: Red Hat OpenShift Development II: Containerizing Applications	437
Ausführliche Wiederholung	438
Praktische Übung: Ausführliche Wiederholung	440
A. Erstellen eines GitHub-Benutzerkontos	455
Erstellen eines GitHub-Benutzerkontos	456
B. Erstellen eines Quay-Benutzerkontos	461
Erstellen eines Quay-Benutzerkontos	462
C. Nützliche Git-Befehle	467
Git-Befehle	468

Dokumentkonventionen



Literaturhinweise

„Verweise“ geben an, wo Sie weitere Informationen zu einem Thema in externen Dokumentationen finden können.



Anmerkung

„Hinweise“ sind Tipps, Tastenkombinationen oder alternative Ansätze für die vorliegende Aufgabe. Wenn Sie einen Hinweis ignorieren, hat dies normalerweise keine negativen Konsequenzen. Allerdings können Hinweise helfen, einen Vorgang zu optimieren.



Wichtig

In den Feldern „Wichtig“ werden Details hervorgehoben, die andernfalls leicht übersehen werden könnten: Konfigurationsänderungen, die nur die aktuelle Sitzung betreffen, oder Dienste, die neu gestartet werden müssen, bevor ein Update angewendet werden kann. Wenn Sie ein Feld mit der Bezeichnung „Wichtig“ ignorieren, führt dies nicht zu Datenverlust, kann jedoch Irritationen und Frustration hervorrufen.



Warnung

„Warnungen“ dürfen nicht ignoriert werden. Ignorierte Warnungen führen mit großer Wahrscheinlichkeit zu einem Datenverlust.

Einführung

Red Hat OpenShift Development II: Containerizing Applications

Red Hat OpenShift Container Platform, das auf der Container-Technologie und Kubernetes basiert, bietet Entwicklern eine bewährte Unternehmenslösung für die Entwicklung und Bereitstellung von containerisierten Softwareanwendungen.

Red Hat OpenShift Development II: Containerizing Applications (DO288) ist der zweite Kurs im Bereich OpenShift-Entwicklung und vermittelt Kursteilnehmern die Vorgehensweise zum Entwerfen, Erstellen und Bereitstellen von containerisierten Softwareanwendungen auf OpenShift-Clustern. Unabhängig davon, ob Sie systemeigene Container-Anwendungen entwerfen oder vorhandene Anwendungen migrieren: Dieser Kurs bietet eine praxisorientierte Schulung zur Steigerung der Produktivität von Entwicklern mit Red Hat OpenShift Container Platform.

Lerninhalte

- Entwerfen, Erstellen und Bereitstellen von containerisierten Softwareanwendungen auf OpenShift-Clustern

Zielgruppe

- Software-Entwickler
- Software-Architekten

Voraussetzungen

- Abschluss des Kurses „Introduction to Containers, Kubernetes, and Red Hat OpenShift (DO180)“ oder gleichwertige Kenntnisse.
- RHCSA oder höher ist hilfreich für die Navigation und Verwendung der Befehlszeile, jedoch nicht obligatorisch.

Informationen zur Kursumgebung

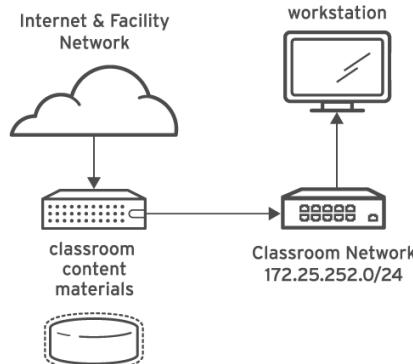


Abbildung 0.1: Kursumgebung

In diesem Kurs wird **workstation** als primäres Computersystem für praktische Übungen verwendet. Das ist ein virtueller Rechner (VM) mit dem Namen **workstation.lab.example.com**.

Alle Kursteilnehmer-Rechner verfügen über das standardmäßige Benutzerkonto **student** mit dem Passwort **student**. Das **root**-Passwort für alle Kursteilnehmer-Systeme lautet **redhat**.

Kursraum-Rechner

Rechnername	IP-Adressen	Rolle
content.example.com, materials.example.com, classroom.example.com	172.25.252.254, 172.25.253.254, 172.25.254.254	Utility Server des Kursraums
workstation.lab.example.com	172.25.250.254, 172.25.252.1	Grafische Workstation für Teilnehmer

Im Kursraum bieten verschiedene Systeme Unterstützung. Die beiden Server **content.example.com** und **materials.example.com** dienen als Quelle für Software- und Übungsmaterialien für praktische Übungen. Informationen zur Verwendung dieser Server finden Sie in der Anleitung der jeweiligen Übungen.

Die Kursteilnehmer greifen über den Rechner **workstation** auf einen gemeinsam genutzten OpenShift-Cluster zu, der extern in AWS gehostet wird. Die Kursteilnehmer verfügen über keine Administratorberechtigungen für den Cluster, dies ist jedoch nicht erforderlich, um den DO288-Inhalt abzuschließen.

Den Kursteilnehmern wird ein Benutzerkonto in einem freigegebenen OpenShift 4-Cluster bereitgestellt, wenn sie ihre Umgebungen in der Red Hat Online Learning-Schnittstelle bereitstellen. Cluster-Informationen wie API-Endpunkt und Cluster-ID sowie ihr Benutzername und Kennwort werden ihnen bei der Bereitstellung ihrer Umgebung angezeigt.

Einführung

Die Kursteilnehmer haben auch Zugriff auf einen MySQL- und einen Nexus-Server, der entweder vom OpenShift-Cluster oder von AWS gehostet wird. Praktische Aktivitäten in diesem Kurs enthalten Anweisungen für den Zugriff auf diese Server, sofern erforderlich.

Praktische Aktivitäten in DO288 erfordern auch, dass die Kursteilnehmer über persönliche Benutzerkonten bei zwei öffentlichen, kostenlosen Internetservices verfügen: GitHub und Quay.io. Die Kursteilnehmer müssen diese Benutzerkonten einrichten, wenn sie noch nicht über sie verfügen (siehe Anhang), und ihren Zugriff überprüfen, indem sie sich bei diesen Services anmelden, bevor sie den Kurs beginnen.

Steuerung Ihrer Systeme

Kursteilnehmern werden Remote-Computer in einem Red Hat Online Learning-Kursraum zugewiesen. Der Zugriff darauf erfolgt über eine unter rol.redhat.com [<http://rol.redhat.com>] gehostete Webanwendung. Kursteilnehmer sollten sich mithilfe ihrer Anmeldedaten für das Red Hat Customer Portal auf dieser Website anmelden.

Steuern der virtuellen Rechner

Die virtuellen Rechner in Ihrer Kursumgebung werden über eine Webseite gesteuert. Der Status jedes virtuellen Rechners im Kursraum wird auf der unter dem Tab **Online Lab** befindlichen Seite angezeigt.

Rechnerstatus

VM-Status	Beschreibung
STARTING	Der virtuelle Rechner wird hochgefahren.
STARTED	Der virtuelle Rechner wird ausgeführt und ist verfügbar (oder, falls noch hochgefahren wird, wird es bald sein.)
STOPPING	Der virtuelle Rechner wird heruntergefahren.
STOPPED	Der virtuelle Rechner ist vollständig heruntergefahren. Beim Starten fährt der virtuelle Rechner in denselben Status hoch, in dem er sich vor dem Herunterfahren befand. (Die Disk wurde nicht gelöscht.)
PUBLISHING	Der virtuelle Rechner wird erstmalig erstellt.
WAITING_TO_START	Der virtuelle Rechner wartet auf den Start anderer virtueller Rechner.

In Abhängigkeit des Status eines Rechners steht eine Auswahl der folgenden Aktionen zur Verfügung.

Aktionen für Kursumgebung/Rechner

Schaltfläche oder Aktion	Beschreibung
PROVISION LAB	Erstellt den ROL-Kursraum. Hiermit werden sämtliche für den Kursraum erforderlichen virtuellen Rechner erstellt und gestartet. Dies dauert ggf. mehrere Minuten.

Schaltfläche oder Aktion	Beschreibung
DELETE LAB	Entfernt den ROL-Kursraum. Hiermit werden alle virtuellen Rechner im Kursraum entfernt. Achtung: Alle auf den Disks gespeicherten Arbeiten gehen verloren.
START LAB	Startet alle virtuellen Rechner im Kursraum.
SHUTDOWN LAB	Hält alle virtuellen Rechner im Kursraum an.
OPEN CONSOLE	Öffnet einen neuen Tab im Browser und stellt eine Verbindung zwischen Konsole und virtuellem Rechner her. Kursteilnehmer können sich direkt beim virtuellen Rechner anmelden und Befehle ausführen. In den meisten Fällen sollten sich die Kursteilnehmer beim virtuellen Rechner workstation anmelden und ssh verwenden, um mit anderen virtuellen Rechnern eine Verbindung herzustellen.
ACTION → Start	Startet den virtuellen Rechner (d. h. schaltet ihn ein).
ACTION → Shutdown	Fährt den virtuellen Rechner ordnungsgemäß herunter, damit die Disk-Inhalte nicht verloren gehen.
ACTION → Power Off	Erzwingt ein Herunterfahren des virtuellen Rechners und behält die Inhalte seiner Disk bei. Dies entspricht der Stromabschaltung bei einem physischen Rechner.
ACTION → Reset	Erzwingt das Herunterfahren des virtuellen Rechners und setzt die Disk in den Ursprungszustand zurück. Achtung: Sämtliche auf der Disk gespeicherte Arbeit geht verloren.

Klicken Sie zu Beginn einer Übung, sofern Sie angewiesen wurden, einen einzelnen Knoten eines virtuellen Rechners zurückzusetzen, nur für den bestimmten virtuellen Rechner auf ACTION → Reset.

Klicken Sie zu Beginn einer Übung, sofern Sie angewiesen wurden, alle virtuellen Rechner zurückzusetzen, auf ACTION → Reset.

Wenn Sie die Kursumgebung auf ihren ursprünglichen Zustand beim Start des Kurses zurücksetzen möchten, können Sie auf DELETE LAB klicken, um die gesamte Kursumgebung zu entfernen.

Klicken Sie nach dem Löschen des Labs auf PROVISION LAB, um einen neuen Satz von Kurssystemen bereitzustellen.



Warnung

Der Vorgang DELETE LAB kann nicht rückgängig gemacht werden. Die von Ihnen bis zu diesem Zeitpunkt in der Kursumgebung vorgenommene Arbeit geht verloren.

Der Autostop-Timer

Die Registrierung bei Red Hat Online Learning ermöglicht Kursteilnehmern eine bestimmte Menge Zeit am Computer. Für den sparsamen Umgang mit der vorgegebenen Computerzeit verfügt der ROL-Kursraum über einen verknüpften Zählvorgang, der die Kursumgebung herunterfährt, wenn der Timer abgelaufen ist.

Klicken Sie zum Anpassen des Timers auf **MODIFY**, damit das Dialogfeld **New Autostop Time** angezeigt wird. Legen Sie die Anzahl der Stunden und Minuten fest, bis der Kursraum automatisch gehalten wird. Beachten Sie, dass die maximale Zeit zehn Stunden beträgt. Klicken Sie auf **ADJUST TIME**, um diese Änderung auf die Timer-Einstellungen anzuwenden.

Landessprachliche Anpassung

Sprachauswahl auf Benutzerbasis

Ihre Benutzer bevorzugen für ihre Desktop-Umgebung eventuell eine andere Sprache als die Standardsprache des Systems. Für ihr Benutzerkonto möchten sie möglicherweise auch ein anderes Tastaturlayout oder eine andere Eingabemethode verwenden.

Spracheinstellungen

In der GNOME-Desktop-Umgebung wird der Benutzer möglicherweise bei der ersten Anmeldung aufgefordert, seine bevorzugte Sprache und Eingabemethode einzustellen. Ist dies nicht der Fall, ist die Anwendung Region & Language für den einzelnen Benutzer der einfachste Weg, die bevorzugte Sprache und Eingabemethode anzupassen.

Sie können diese Anwendung auf zwei Arten starten. Sie können den Befehl `gnome-control-center region` über ein Terminal ausführen. Wählen Sie alternativ auf der oberen Leiste im Systemmenü in der rechten Ecke die Schaltfläche für die Einstellungen (das Symbol mit dem Kreuzschlitz-Schraubendreher und Schraubenschlüssel) im unteren linken Bereich des Menüs aus.

Wählen Sie in dem sich öffnenden Fenster Region & Language aus. Klicken Sie auf das Feld **Language**, und wählen Sie aus der daraufhin angezeigten Liste die bevorzugte Sprache aus. Dadurch wird auch die Einstellung für **Formats** auf die Standardwerte dieser Sprache aktualisiert. Bei Ihrer nächsten Anmeldung werden die Änderungen vollständig wirksam.

Diese Einstellungen wirken sich auf die GNOME-Desktop-Umgebung sowie auf alle Anwendungen, beispielsweise `gnome-terminal` aus, die innerhalb der Umgebung gestartet werden. Standardmäßig werden sie jedoch nicht auf dieses Benutzerkonto angewendet, wenn über eine ssh-Anmeldung von einem Remote-System oder über eine textbasierte Anmeldung von einer virtuellen Konsole (beispielsweise `tty5`) darauf zugegriffen wird.



Anmerkung

Sie können festlegen, dass Ihre Shell-Umgebung dieselbe LANG-Einstellung wie Ihre grafische Umgebung verwendet, selbst wenn Sie sich über eine textbasierte virtuelle Konsole oder über ssh anmelden. Hierzu kann beispielsweise Code ähnlich dem folgenden in Ihrer `~/.bashrc`-Datei platziert werden. Der Code im folgenden Beispiel stellt die Sprache für eine Textanmeldung so ein, dass sie mit der zurzeit für die GNOME Desktop-Umgebung des Benutzers konfigurierten Sprache übereinstimmt:

```
i=$(grep 'Language=' /var/lib/AccountsService/users/${USER} \
| sed 's/Language=//')
if [ "$i" != "" ]; then
    export LANG=$i
fi
```

Japanisch, Koreanisch, Chinesisch und andere Sprachen mit nicht lateinischem Zeichensatz werden in textbasierten virtuellen Konsolen eventuell nicht einwandfrei angezeigt.

Einführung

Durch Festlegen der Variablen LANG in der Befehlszeile kann für einzelne Befehle die Verwendung einer anderen Sprache festgelegt werden:

```
[user@host ~]$ LANG=fr_FR.utf8 date  
jeu. avril 25 17:55:01 CET 2019
```

Bei den nachfolgenden Befehlen erfolgt die Ausgabe wieder in der Standardsprache des Systems. Der Befehl `Locale` kann verwendet werden, um den aktuellen Wert von LANG und andere relevante Umgebungsvariablen zu bestimmen.

Einstellungen der Eingabemethode

GNOME 3 in Red Hat Enterprise Linux 7 oder höher verwendet automatisch das Auswahlsystem der Eingabemethode IBus, wodurch sich die Tastaturlayouts und Eingabemethoden schnell wechseln lassen.

Die Anwendung Region & Language kann auch zum Aktivieren alternativer Eingabemethoden verwendet werden. Im Anwendungsfenster von Region & Language zeigt das Feld **Input Sources** an, welche Eingabemethoden derzeit verfügbar sind. Standardmäßig ist eventuell **English (US)** die einzige verfügbare Methode. Markieren Sie **English (US)**, und klicken Sie auf das **Tastatur**-Symbol, um das aktuelle Tastaturlayout anzuzeigen.

Um eine weitere Eingabemethode hinzuzufügen, klicken Sie auf die Schaltfläche + im linken unteren Bereich des Fensters **Input Sources**. Das Fenster **Add an Input Source** wird angezeigt. Wählen Sie Ihre Sprache und anschließend die bevorzugte Eingabemethode oder das Tastaturlayout aus.

Wenn mehr als eine Eingabemethode konfiguriert ist, kann der Benutzer rasch zwischen diesen wechseln, indem er **Super+Space** (manchmal auch **Windows+Space**) eingibt. Auf der oberen GNOME-Leiste wird ein **Statusindikator** angezeigt, der über zwei Funktionen verfügt: Er gibt an, welche Ausgabemethode aktiv ist, und fungiert als Menü, das zum Wechseln zwischen Eingabemethoden oder zur Auswahl von erweiterten komplexeren Eingabemethoden verwendet werden kann.

Einige der Methoden sind mit Zahnrad-Symbolen gekennzeichnet. Diese verfügen über erweiterte Konfigurationsoptionen und Funktionen. Beispielsweise kann der Benutzer mit der japanischen Eingabemethode **Japanese (Kana Kanji)** Text im lateinischen Zeichensatz vorab bearbeiten und mit den Tasten **Pfeil nach unten** und **Pfeil nach oben** die zu verwendenden Zeichen auswählen.

Englischsprachige Benutzer in den USA finden dies ggf. ebenfalls hilfreich. Bei der Eingabemethode **English (United States)** lautet beispielsweise das Tastaturlayout **English (international AltGr dead keys)**, das die Taste **AltGr** (oder die rechte **Alt**-Taste) auf einer PC-Tastatur mit 104/105 Tasten als Zusatztaste in Form einer „zweiten Umschalttaste“ sowie als Aktivierungstaste für nicht belegte Tasten zur Eingabe zusätzlicher Zeichen behandelt. Zur Verfügung stehen auch Dvorak und andere Tastaturlayouts.



Anmerkung

In der GNOME Desktop-Umgebung können alle Unicode-Zeichen eingegeben werden, sofern Sie den Unicode-Codepoint oder Unicode-Zahlenwert des Zeichens kennen. Drücken Sie **Strg+Umschalt+U**, und geben Sie dann den Codepoint ein. Nach dem Drücken von **Strg+Umschalt+U** erscheint ein unterstrichenes u, das angibt, dass das System auf die Eingabe des Unicode-Codepoints wartet.

Beispielsweise hat der kleine griechische Buchstabe Lambda den Codepoint U +03BB und wird durch Eingabe von **Strg+Umschalt+U**, anschließend **03BB** und Drücken der Eingabetaste eingegeben.

Einstellungen der Standardsprache des Systems

Die Standardsprache des Systems ist US-Englisch mit den Unicode-Zeichen der UTF-8-Codierung (`en_US.utf8`). Dies lässt sich aber während oder nach der Installation ändern.

In der Befehlszeile kann der `root`-Benutzer die systemweiten Ländereinstellungen mit dem Befehl `localectl` ändern. Wenn `localectl` ohne Argumente ausgeführt wird, werden die aktuellen Ländereinstellungen des Systems angezeigt.

Führen Sie zum Einstellen der systemweiten Standardsprache den Befehl `localectl set-locale LANG=locale` aus, wobei `locale` der entsprechende Wert für die `LANG`-Umgebungsvariable aus der Tabelle „Sprachcodereferenz“ in diesem Kapitel ist. Die Änderung wird für die Benutzer bei der nächsten Anmeldung wirksam und wird in der Datei `/etc/locale.conf` gespeichert.

```
[root@host ~]# localectl set-locale LANG=fr_FR.utf8
```

In GNOME kann ein Administrator diese Einstellung ändern, indem er in Region & Language auf die Schaltfläche des **Anmeldebildschirms** in der oberen rechten Ecke des Fensters klickt. Das Ändern der **Sprache** des grafischen Anmeldebildschirms wirkt sich auch auf die Einstellungen der Standardsprache des Systems aus, die in der Konfigurationsdatei `/etc/locale.conf` gespeichert sind.



Wichtig

Textbasierte virtuelle Konsolen wie `ttys` sind hinsichtlich der Schriftarten, die sie anzeigen können, eingeschränkt als Terminals in einer virtuellen Konsole, die in einer grafischen Umgebung ausgeführt wird, oder als Pseudoterminals für `ssh`-Sitzungen. Japanische, koreanische und chinesische Zeichen beispielsweise werden in einer textbasierten virtuellen Konsole eventuell nicht richtig angezeigt. Daher sollten Sie in Betracht ziehen, Englisch oder eine andere Sprache mit einem lateinischen Zeichensatz als systemweite Standardeinstellung zu verwenden.

Gleichermaßen sind textbasierte virtuelle Konsolen bei den von ihnen unterstützten Eingabemethoden eingeschränkt. Dies wird separat über die grafische Desktopumgebung verwaltet. Die verfügbaren globalen Eingabeeinstellungen werden sowohl für textbasierte virtuelle Konsolen als auch für die grafische Umgebung anhand von `localectl` konfiguriert. Weitere Informationen finden Sie auf den Manpages `localectl(1)` und `vconsole.conf(5)`.

Sprachpakete

Mit speziellen RPM-Paketen, die als *langpacks* bezeichnet werden, werden Sprachpakete installiert, die Unterstützung für bestimmte Sprachen hinzufügen. Diese Sprachpakete nutzen Abhängigkeiten, um zusätzliche RPM-Pakete, die Lokalisierungen, Verzeichnisse und Übersetzungen für andere Softwarepakete enthalten, automatisch auf Ihrem System zu installieren.

Verwenden Sie `yum list langpacks-*` zum Auflisten der Sprachpakete, die installiert sind und möglicherweise installiert werden:

```
[root@host ~]# yum list langpacks-*
Updating Subscription Management repositories.
Updating Subscription Management repositories.
Installed Packages
langpacks-en.noarch      1.0-12.el8          @AppStream
Available Packages
langpacks-af.noarch       1.0-12.el8          rhel-8-for-x86_64-appstream-rpms
langpacks-am.noarch       1.0-12.el8          rhel-8-for-x86_64-appstream-rpms
langpacks-ar.noarch       1.0-12.el8          rhel-8-for-x86_64-appstream-rpms
langpacks-as.noarch       1.0-12.el8          rhel-8-for-x86_64-appstream-rpms
langpacks-ast.noarch      1.0-12.el8          rhel-8-for-x86_64-appstream-rpms
...output omitted...
```

Um Sprachunterstützung hinzuzufügen, installieren Sie das entsprechende Sprachpaket. Mit dem folgenden Befehl wird beispielsweise Unterstützung für Französisch hinzugefügt:

```
[root@host ~]# yum install langpacks-fr
```

Mit `yum repoquery --what supplements` können Sie bestimmen, welche RPM-Pakete durch ein Sprachpaket installiert werden können:

```
[root@host ~]# yum repoquery --what supplements langpacks-fr
Updating Subscription Management repositories.
Updating Subscription Management repositories.
Last metadata expiration check: 0:01:33 ago on Wed 06 Feb 2019 10:47:24 AM CST.
glibc-langpack-fr-0:2.28-18.el8.x86_64
gnome-getting-started-docs-fr-0:3.28.2-1.el8.noarch
hunspell-fr-0:6.2-1.el8.noarch
hyphen-fr-0:3.0-1.el8.noarch
libreoffice-langpack-fr-1:6.0.6.1-9.el8.x86_64
man-pages-fr-0:3.70-16.el8.noarch
mythes-fr-0:2.3-10.el8.noarch
```



Wichtig

Sprachpakete nutzen *schwache Abhängigkeiten* für RPM, damit Zusatzpakete nur installiert werden, wenn das Kernpaket, das sie benötigt, ebenfalls installiert ist.

Wenn beispielsweise *langpacks-fr* wie in den vorherigen Beispielen gezeigt installiert wird, wird das Paket *mythes-fr* nur installiert, wenn auch der Thesaurus *mythes* auf dem System installiert ist.

Wenn *mythes* anschließend auf dem System installiert wird, wird das Paket *mythes-fr* aufgrund der schwachen Abhängigkeit des bereits installierten Pakets *langpacks-fr* ebenfalls automatisch installiert.



Literaturhinweise

Manpages `locale(7)`, `localectl(1)`, `locale.conf(5)`, `vconsole.conf(5)`, `unicode(7)` und `utf-8(7)`

Konvertierungen zwischen den Namen der X11-Layouts der grafischen Desktopumgebung und deren Namen in `localectl` befinden sich in der Datei `/usr/share/X11/xkb/rules/base.lst`.

Sprachcodereferenz



Anmerkung

Diese Tabelle enthält möglicherweise nicht alle auf Ihrem System verfügbaren Sprachpakete. Verwenden Sie `yum info langpacks-SUFFIX`, um weitere Informationen zum jeweiligen Sprachpaket abzurufen.

Sprachcodes

Sprache	Suffix für Sprachpakete	\$LANG-Wert
Englisch (US)	en	en_US.utf8
Assamesisch	as	as_IN.utf8
Bengalisch	bn	bn_IN.utf8
Chinesisch (vereinfacht)	zh_CN	zh_CN.utf8
Chinesisch (traditionell)	zh_TW	zh_TW.utf8
Französisch	fr	fr_FR.utf8
Deutsch	de	de_DE.utf8
Gujarati	gu	gu_IN.utf8
Hindi	hi	hi_IN.utf8

Sprache	Suffix für Sprachpakete	\$LANG-Wert
Italienisch	it	it_IT.utf8
Japanisch	ja	ja_JP.utf8
Kanaresisch	kn	kn_IN.utf8
Koreanisch	ko	ko_KR.utf8
Malayalam	ml	ml_IN.utf8
Marathisch	mr	mr_IN.utf8
Odia	or	or_IN.utf8
Portugiesisch (Brasilien)	pt_BR	pt_BR.utf8
Punjabi	pa	pa_IN.utf8
Russisch	ru	ru_RU.utf8
Spanisch	es	es_ES.utf8
Tamilisch	ta	ta_IN.utf8
Telugu	te	te_IN.utf8

Kapitel 1

Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster

Ziel

Bereitstellen von Anwendungen mithilfe verschiedener Methoden für das Packen von Anwendungen auf einem OpenShift-Cluster und Verwalten der zugehörigen Ressourcen

Ziele

- Beschreiben der Architektur und neuen Funktionen von OpenShift 4
- Bereitstellen einer Anwendung auf dem Cluster aus einem Dockerfile mit der CLI
- Bereitstellen einer Anwendung über ein Container-Image und Verwalten der zugehörigen Ressourcen mit der Web Console
- Bereitstellen einer Anwendung über den Quellcode und Verwalten der zugehörigen Ressourcen mithilfe der Befehlszeilenschnittstelle

Abschnitte

- Einführung in OpenShift 4 (und Test)
- Bereitstellen einer Anwendung auf einem OpenShift-Cluster (und angeleitete Übung)
- Verwalten von Anwendungen mit der Web Console (und angeleitete Übung)
- Verwalten von Anwendungen mit der CLI (und angeleitete Übung)

Praktische Übung

Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster

Einführung in OpenShift Container Platform 4

Ziele

Nach Abschluss dieses Abschnitts sollten Sie in der Lage sein, die Architektur und die neuen Funktionen von OpenShift Container Platform 4 zu beschreiben.

Architektur von OpenShift Container Platform 4

Red Hat OpenShift Container Platform 4 (RHOCOP 4) besteht aus einer Reihe modularer Komponenten und Services, die auf Red Hat CoreOS und Kubernetes basieren. OpenShift fügt Platform-as-a-Service (PaaS)-Funktionen wie Remote-Management, erhöhte Sicherheit, Überwachung und Auditing, Lifecycle-Management für Anwendungen und Self-Service-Schnittstellen für Entwickler hinzu. Es bietet Orchestrierungsservices und vereinfacht Deployment, Verwaltung und Skalierung von containerisierten Anwendungen.

Ein OpenShift-Cluster kann auf die gleiche Weise wie jeder andere Kubernetes-Cluster verwaltet werden, kann aber auch mit den OpenShift-Management-Tools, z. B. der Befehlszeilenschnittstelle oder der Web Console, verwaltet werden. Diese zusätzlichen Tools ermöglichen produktivere Workflows und vereinfachen alltägliche Verwaltungsaufgaben wesentlich.

Ein Hauptvorteil der Verwendung von OpenShift besteht darin, dass mehrere Knoten verwendet werden, um Ausfallsicherheit und Skalierbarkeit der verwalteten Anwendungen sicherzustellen. OpenShift bildet einen Cluster von Knoten-Servern, die in Containern ausgeführt und zentral von einer Reihe von Master-Servern verwaltet werden. Ein einzelner Host kann als Master und als Knoten fungieren, aber Sie sollten diese Rollen aus Gründen einer höheren Stabilität und besseren Hochverfügbarkeit normalerweise trennen.

Das folgende Diagramm zeigt die allgemeine logische Übersicht über die Architektur von OpenShift Container Platform 4.

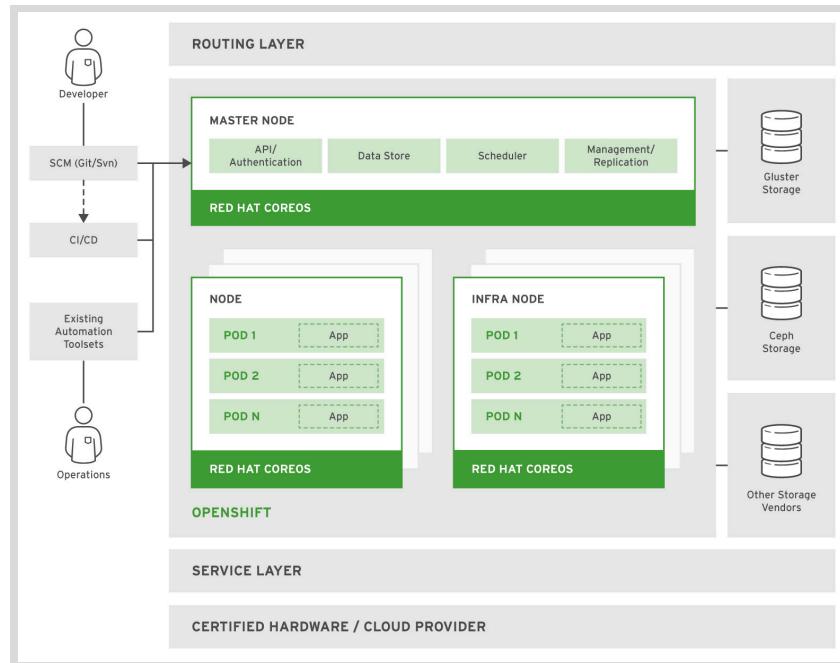


Abbildung 1.1: OpenShift 4-Architektur

Das folgende Diagramm zeigt den OpenShift Container Platform-Stack.

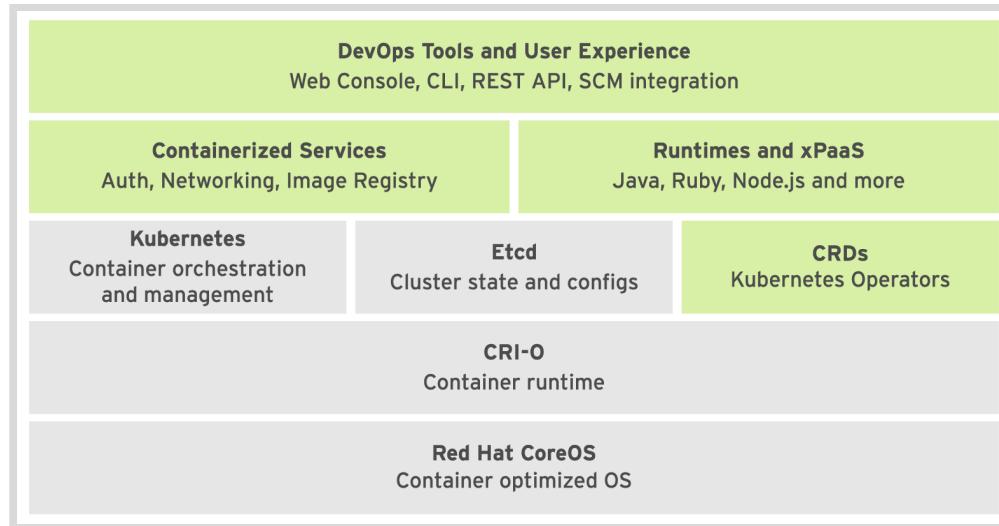


Abbildung 1.2: OpenShift-Komponenten-Stack

Die grundlegende durch Red Hat integrierte und erweiterte Container-Infrastruktur ist von unten nach oben und von links nach rechts dargestellt:

Red Hat CoreOS

Red Hat CoreOS ist das Basisbetriebssystem, auf dem OpenShift ausgeführt wird. Red Hat CoreOS ist eine Linux-Distribution, die auf die Bereitstellung eines unveränderlichen Betriebssystems für die Container-Ausführung ausgerichtet ist.

CRI-O

CRI-O ist eine Implementierung des Kubernetes Container Runtime Interface (CRI) für die Verwendung von mit Open Container Initiative (OCI) kompatiblen Laufzeiten. CRI-O kann

Kapitel 1 | Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster

jede Container-Laufzeit verwenden, die der CRI entspricht: `runc` (wird vom Docker-Service verwendet) oder `rkt` (von CoreOS).

Kubernetes

Kubernetes verwaltet einen Cluster mit physischen oder virtuellen Hosts, die Container ausführen. Es verwendet *Ressourcen*, die aus mehreren Ressourcen bestehende Multicontainer-Anwendungen und deren Kommunikation untereinander beschreiben.

etcd

`etcd` ist ein verteilter Schlüssel-Wert-Speicher, der von Kubernetes zum Speichern der Konfigurations- und Statusinformationen der Container und anderer Ressourcen im Kubernetes-Cluster verwendet wird.

Benutzerdefinierte Ressourcendefinitionen (CRDs)

Benutzerdefinierte Ressourcendefinitionen (CRDs) sind Ressourcentypen, die in `etcd` gespeichert und von Kubernetes verwaltet werden. Diese Ressourcentypen bilden den Status und die Konfiguration aller von OpenShift verwalteten Ressourcen.

Containerisierte Services

Containerisierte Services nehmen viele PaaS-Infrastrukturfunktionen wahr, wie Netzwerk und Autorisierung. RHOPC verwendet die grundlegende Container-Infrastruktur von Kubernetes und die zugrunde liegende Container-Laufzeit für die meisten internen Funktionen. Das heißt, die meisten internen RHOPC-Services werden als von Kubernetes orchestrierte Container ausgeführt.

Laufzeiten und xPaaS

Laufzeiten und xPaaS sind grundlegende, gebrauchsfertige Container-Images für Entwickler, jedes ist mit einer bestimmten Laufzeitsprache oder einer Datenbank vorkonfiguriert. Das xPaaS-Angebot besteht aus einer Reihe von Basis-Images für Red Hat-Middleware-Produkte, beispielsweise JBoss EAP und ActiveMQ. *Red Hat OpenShift Application Runtimes (RHOAR)* sind Laufzeiten, die für native Cloud-Anwendungen in OpenShift optimiert wurden. Die verfügbaren Anwendungslaufzeiten sind Red Hat JBoss EAP, OpenJDK, Thorntail, Eclipse Vert.x, Spring Boot und Node.js.

DevOps-Tools und Benutzererlebnis

DevOps-Tools und Benutzererlebnis: RHOPC bietet Management-Tools für die Web-UI und die CLI zur Verwaltung von Benutzeroberflächen und RHOPC-Services. Die Web-UI- und CLI-Tools von OpenShift sind aus REST-APIs erstellt, die von externen Tools wie IDEs und CI-Plattformen verwendet werden können.

In der folgenden Tabelle sind einige der am häufigsten verwendeten Begriffe für die Arbeit mit OpenShift aufgeführt.

OpenShift-Terminologie

Begriff	Definition
Knoten	Ein Server, der Anwendungen in einem OpenShift-Cluster hostet.
Master-Knoten	Ein Knoten-Server, der die Kontrollsicht in einem OpenShift-Cluster verwaltet. Master-Knoten stellen grundlegende Cluster-Services wie APIs oder Controller bereit.
Worker-Knoten	Worker-Knoten werden auch als Server-Knoten bezeichnet und führen Workloads für den Cluster aus. Anwendungs-Pods werden auf Worker-Knoten geplant.

Begriff	Definition
Ressource	Ressourcen sind jegliche Art von Komponentendefinitionen, die von OpenShift verwaltet werden. Ressourcen enthalten die Konfiguration der verwalteten Komponente (beispielsweise die einem Knoten zugewiesene Rolle) und den aktuellen Status der Komponente (beispielsweise, ob der Knoten verfügbar ist).
Controller	Ein Controller ist eine OpenShift-Komponente, die Ressourcen überwacht und Änderungen vornimmt, um den aktuellen Status in den gewünschten Status zu ändern.
Label	Ein Schlüssel-Wert-Paar, das einer beliebigen OpenShift-Ressource zugewiesen werden kann. Selektoren verwenden Bezeichnungen zum Filtern geeigneter Ressourcen für die Planung und andere Vorgänge.
Namespace oder Projekt	Ein Bereich für OpenShift-Ressourcen und -Prozesse, damit Ressourcen mit demselben Namen in unterschiedlichen Kontexten verwendet werden können.
Konsole	Eine von OpenShift bereitgestellte Web-Benutzeroberfläche, über die Entwickler und Administratoren Cluster-Ressourcen verwalten können.



Anmerkung

Die neuesten Versionen von OpenShift implementieren viele Controller als Operatoren. Operatoren sind Kubernetes-Plugin-Komponenten, die auf Cluster-Ereignisse reagieren und den Status von Ressourcen steuern können. Operatoren und Operator Framework werden in diesem Dokument nicht behandelt.

Neue Funktionen in RHOC 4

RHOC 4 ist eine tiefgreifende Änderung gegenüber früheren Versionen. Neben der Abwärtskompatibilität mit früheren Versionen enthält es neue Funktionen, wie:

- CoreOS als Standardbetriebssystem für alle Knoten mit einer unveränderlichen Infrastruktur, die für Container optimiert ist.
- Ein neues Cluster-Installationsprogramm, das Installation und Aktualisierung der Master- und Worker-Knoten im Cluster vereinfacht.
- Eine Selbstverwaltungsplattform, die Cluster-Updates und -Wiederherstellungen ohne Unterbrechung automatisch durchführen kann.
- Eine neu gestaltete Web Console, die auf dem Rollenkonzept basiert und auf Plattformadministratoren und -entwickler ausgerichtet ist.
- Ein Operator-SDK zum Erstellen, Testen und Packen von Operatoren.

Beschreiben von OpenShift-Ressourcentypen

Als Entwickler arbeiten Sie in OpenShift mit vielen unterschiedlichen Arten von Ressourcentypen. Diese Ressourcen können mithilfe einer YAML- bzw. JSON-Datei oder mithilfe der Management-Tools von OpenShift erstellt und konfiguriert werden:

Pods (Pod)

Sammlungen von Containern, die dieselben Ressourcen verwenden, zum Beispiel IP-Adressen und Volumes für persistenten Storage. Pods sind die grundlegende Arbeitseinheit für OpenShift.

Services (SVC)

Spezifische IP-/Port-Kombinationen, die den Zugriff auf einen Pool von Pods ermöglichen. Services stellen Verbindungen zwischen Clients und Pods in der Regel mit dem Roundrobin-Verfahren her.

Replication Controller (RC)

OpenShift-Ressourcen, die definieren, wie Pods auf unterschiedliche Knoten repliziert werden (horizontal skaliert). Replication Controller sind ein grundlegender OpenShift-Service, der eine hohe Verfügbarkeit für Pods und Container bietet.

Persistente Volumes (PV)

Speicherbereiche, die von Pods verwendet werden sollen.

Anforderung für ein persistentes Volume (PVC)

Anforderungen auf Storage durch einen Pod. Eine PVC verbindet ein PV mit einem Pod, sodass die Container ihn verwenden können, üblicherweise durch Mounten des Storages in das Dateisystem des Containers.

ConfigMaps (CM)

Einen Satz von Schlüsseln und Werten, die von anderen Ressourcen verwendet werden können. ConfigMaps und Secrets werden normalerweise verwendet, um Konfigurationswerte zu zentralisieren, die von mehreren Ressourcen verwendet werden. Secrets unterscheiden sich von ConfigMaps dadurch, dass mit Secrets vertrauliche Daten (normalerweise verschlüsselt) gespeichert werden und der Zugriff darauf auf weniger autorisierte Benutzer beschränkt ist.

Deploymentkonfigurationen (DC)

Eine Reihe von Containern, die in einem Pod enthalten sind, sowie die zu verwendenden Deploymentstrategien. Eine DC bietet auch einen grundlegenden, aber erweiterbaren Workflow für die kontinuierliche Bereitstellung.

Build-Konfigurationen (BC)

Ein Prozess, der im OpenShift-Projekt ausgeführt werden soll. Die *Source-to-Image*-Funktion (S2I) von OpenShift verwendet BuildConfigs, um ein Container-Image aus Anwendungsquellcode zu erstellen, der in einem Git-Repository gespeichert ist. Eine BC zusammen mit einer DC stellt Workflows für eine grundlegende, jedoch erweiterbare kontinuierliche Integration und Bereitstellung bereit.

Routen

DNS-Hostnamen, die vom OpenShift-Router als Eingangspunkt für verschiedene auf dem Cluster bereitgestellte Anwendungen und Mikroservices erkannt werden.

Image-Streams (IS)

Ein Image-Stream und seine Tags bieten eine Abstraktion für den Verweis auf Container-Images aus OpenShift Container Platform. Mit dem Image-Stream und seinen Tags können Sie nachverfolgen, welche Images verfügbar sind, und sicherstellen, dass Sie das benötigte Image verwenden, auch wenn sich das Image im Repository ändert. Image-Streams enthalten keine tatsächlichen Image-Daten, sondern stellen eine virtuelle Ansicht zusammengehöriger Images dar, ähnlich einem Image-Repository.



Literaturhinweise

Website zur Kubernetes-Dokumentation

<https://kubernetes.io/docs/>

Website der OpenShift-Dokumentation

<https://docs.openshift.com/>

CoreOS Operators and Operator Framework

<https://coreos.com/operators/>

► Quiz

Einführung in OpenShift 4

Wählen Sie die richtigen Antworten auf die folgenden Fragen aus:

Klicken Sie nach Abschluss des Tests auf **CHECK**. Wenn Sie es noch einmal versuchen möchten, klicken Sie auf **RESET**. Klicken Sie auf **SHOW SOLUTION**, damit alle korrekten Antworten angezeigt werden.

► 1. Welche Aussage trifft im Hinblick auf OpenShift-Ergänzungen für Kubernetes zu?

- a. OpenShift bietet Funktionen, mit denen die Anwendungsentwicklung und -bereitstellung auf Kubernetes einfach und effizient gestaltet werden können.
- b. Von OpenShift erstellte Container-Images können nicht allein mit Kubernetes verwendet werden.
- c. Um neue Funktionen zu ermöglichen, verwaltet Red Hat im RHOC-P-Produkt geforkte Versionen von Kubernetes.
- d. Es gibt keine neuen Funktionen für die fortlaufende Integration und Bereitstellung mit RHOC-P, aber Sie können stattdessen externe Tools verwenden.

► 2. Welche Aussage trifft im Hinblick auf persistenten Storage in OpenShift zu?

- a. Entwickler erstellen eine Anforderung für ein persistentes Volume, um einen Cluster-Storage-Bereich anzufordern, den ein Projekt-Pod zum Speichern von Daten verwenden kann.
- b. Eine Anforderung für persistenten Storage stellt einen Storage-Bereich dar, den ein Pod für die Speicherung von Daten anfordern kann, der aber vom Cluster-Administrator bereitgestellt wird.
- c. Eine Anforderung für persistenten Storage ist die Speichermenge, die einem Knoten zugewiesen werden kann, damit Entwickler ermitteln können, wie viel Speicher sie für die Ausführung ihrer Anwendung benötigen.
- d. Eine Anforderung für persistenten Storage ist die Anzahl der CPU-Verarbeitungseinheiten, die einem Anwendungs-Pod zugewiesen werden können. Diese unterliegen einer vom Cluster-Administrator festgelegten Beschränkung.
- e. OpenShift unterstützt persistenten Storage, indem Administratoren den auf dem Knoten verfügbaren Storage direkt auf dem Cluster ausgeführten Anwendungen zuordnen können.

► **3. Welche beiden Aussagen treffen im Hinblick auf OpenShift-Ressourcentypen zu?**

(Wählen Sie zwei Antworten aus.)

- a. Pods sind für die Bereitstellung des eigenen persistenten Storage zuständig.
- b. Alle Pods, die über denselben Replication Controller erstellt werden, müssen auf demselben Knoten ausgeführt werden.
- c. Ein Service ist für die Bereitstellung von IP-Adressen für den externen Zugriff auf Pods zuständig.
- d. Eine Route ist für die Bereitstellung eines Hostnamens für den externen Zugriff auf Pods zuständig.
- e. Ein Replication Controller ist für die Überwachung und Verwaltung der Anzahl der Pods für eine bestimmte Anwendung verantwortlich.

► **4. Welche beiden Aussagen treffen im Hinblick auf die OpenShift 4-Architektur zu?**

(Wählen Sie zwei Antworten aus.)

- a. OpenShift-Knoten können ohne Master verwaltet werden. Die Knoten bilden ein Peer-to-Peer-Netzwerk.
- b. OpenShift-Master verwalten die Pod-Skalierung und planen Pods für die Ausführung auf Knoten.
- c. Auf Master-Knoten in einem Cluster muss Red Hat CoreOS ausgeführt werden.
- d. Auf Master-Knoten in einem Cluster muss Red Hat Enterprise Linux 8 ausgeführt werden.
- e. Auf Master-Knoten in einem Cluster muss Red Hat Enterprise Linux 7 ausgeführt werden.

► Lösung

Einführung in OpenShift 4

Wählen Sie die richtigen Antworten auf die folgenden Fragen aus:

Klicken Sie nach Abschluss des Tests auf **CHECK**. Wenn Sie es noch einmal versuchen möchten, klicken Sie auf **RESET**. Klicken Sie auf **SHOW SOLUTION**, damit alle korrekten Antworten angezeigt werden.

► 1. Welche Aussage trifft im Hinblick auf OpenShift-Ergänzungen für Kubernetes zu?

- a. OpenShift bietet Funktionen, mit denen die Anwendungsentwicklung und -bereitstellung auf Kubernetes einfach und effizient gestaltet werden können.
- b. Von OpenShift erstellte Container-Images können nicht allein mit Kubernetes verwendet werden.
- c. Um neue Funktionen zu ermöglichen, verwaltet Red Hat im RHOC-P-Produkt geforkte Versionen von Kubernetes.
- d. Es gibt keine neuen Funktionen für die fortlaufende Integration und Bereitstellung mit RHOC-P, aber Sie können stattdessen externe Tools verwenden.

► 2. Welche Aussage trifft im Hinblick auf persistenten Storage in OpenShift zu?

- a. Entwickler erstellen eine Anforderung für ein persistentes Volume, um einen Cluster-Storage-Bereich anzufordern, den ein Projekt-Pod zum Speichern von Daten verwenden kann.
- b. Eine Anforderung für persistenten Storage stellt einen Storage-Bereich dar, den ein Pod für die Speicherung von Daten anfordern kann, der aber vom Cluster-Administrator bereitgestellt wird.
- c. Eine Anforderung für persistenten Storage ist die Speichermenge, die einem Knoten zugewiesen werden kann, damit Entwickler ermitteln können, wie viel Speicher sie für die Ausführung ihrer Anwendung benötigen.
- d. Eine Anforderung für persistenten Storage ist die Anzahl der CPU-Verarbeitungseinheiten, die einem Anwendungs-Pod zugewiesen werden können. Diese unterliegen einer vom Cluster-Administrator festgelegten Beschränkung.
- e. OpenShift unterstützt persistenten Storage, indem Administratoren den auf dem Knoten verfügbaren Storage direkt auf dem Cluster ausgeführten Anwendungen zuordnen können.

- **3. Welche beiden Aussagen treffen im Hinblick auf OpenShift-Ressourcentypen zu?**
(Wählen Sie zwei Antworten aus.)
- a. Pods sind für die Bereitstellung des eigenen persistenten Storage zuständig.
 - b. Alle Pods, die über denselben Replication Controller erstellt werden, müssen auf demselben Knoten ausgeführt werden.
 - c. Ein Service ist für die Bereitstellung von IP-Adressen für den externen Zugriff auf Pods zuständig.
 - d. Eine Route ist für die Bereitstellung eines Hostnamens für den externen Zugriff auf Pods zuständig.
 - e. Ein Replication Controller ist für die Überwachung und Verwaltung der Anzahl der Pods für eine bestimmte Anwendung verantwortlich.
- **4. Welche beiden Aussagen treffen im Hinblick auf die OpenShift 4-Architektur zu?**
(Wählen Sie zwei Antworten aus.)
- a. OpenShift-Knoten können ohne Master verwaltet werden. Die Knoten bilden ein Peer-to-Peer-Netzwerk.
 - b. OpenShift-Master verwalten die Pod-Skalierung und planen Pods für die Ausführung auf Knoten.
 - c. Auf Master-Knoten in einem Cluster muss Red Hat CoreOS ausgeführt werden.
 - d. Auf Master-Knoten in einem Cluster muss Red Hat Enterprise Linux 8 ausgeführt werden.
 - e. Auf Master-Knoten in einem Cluster muss Red Hat Enterprise Linux 7 ausgeführt werden.

► Angeleitete Übung

Konfigurieren der Kursumgebung

In dieser Übung konfigurieren Sie die **Workstation** für den Zugriff auf die gesamte Infrastruktur, die in diesem Kurs verwendet wird.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Konfigurieren Ihrer **Workstation** für den Zugriff auf einen OpenShift-Cluster, eine Container-Image-Registry und ein Git-Repository, die im Kurs verwendet werden
- Forken des Repository für die Beispielanwendungen des Kurses auf Ihr persönliches GitHub-Benutzerkonto
- Klonen dieses Repository für die Beispielanwendungen des Kurses von Ihrem persönlichen GitHub-Benutzerkonto auf Ihre **workstation**-VM

Bevor Sie Beginnen

Sie müssen über Folgendes verfügen, um diese Übung durchführen zu können:

- Zugriff auf den DO288-Kurs in der Online-Lernumgebung von Red Hat Training
- Die Verbindungsparameter und ein Entwicklerbenutzerkonto für den Zugriff auf einen OpenShift-Cluster, der von Red Hat Training verwaltet wird
- Ein persönliches, kostenloses GitHub-Benutzerkonto. Lesen Sie die Anweisungen in *Anhang A, Erstellen eines GitHub-Benutzerkontos*, wenn Sie sich bei GitHub registrieren müssen.
- Ein persönliches, kostenloses Quay.io-Benutzerkonto. Lesen Sie die Anweisungen in *Anhang B, Erstellen eines Quay-Benutzerkontos*, wenn Sie sich bei Quay.io registrieren müssen.
- Ein persönliches Zugriffstoken von GitHub.

Anweisungen

Sie müssen alle folgenden Schritte ausführen, bevor Sie mit einer Übung beginnen.

► 1. Bereiten Sie Ihr GitHub-Zugriffstoken vor.

- 1.1. Navigieren Sie mit einem Webbrowser zu <https://github.com>, und authentifizieren Sie sich.
- 1.2. Klicken Sie oben auf der Seite auf Ihr Profilsymbol, wählen Sie das Menü **Settings** aus, und wählen Sie dann im linken Bereich der Seite **Developer settings** aus.

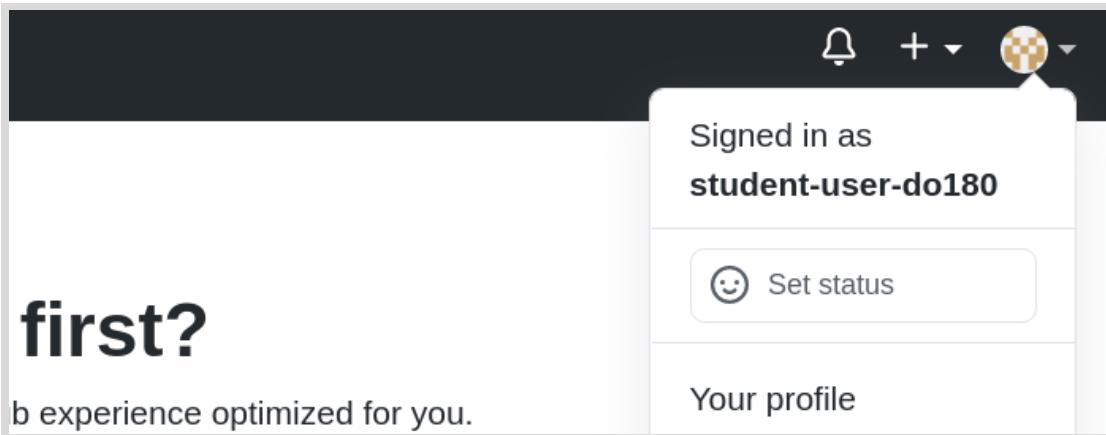


Abbildung 1.3: Benutzermenü

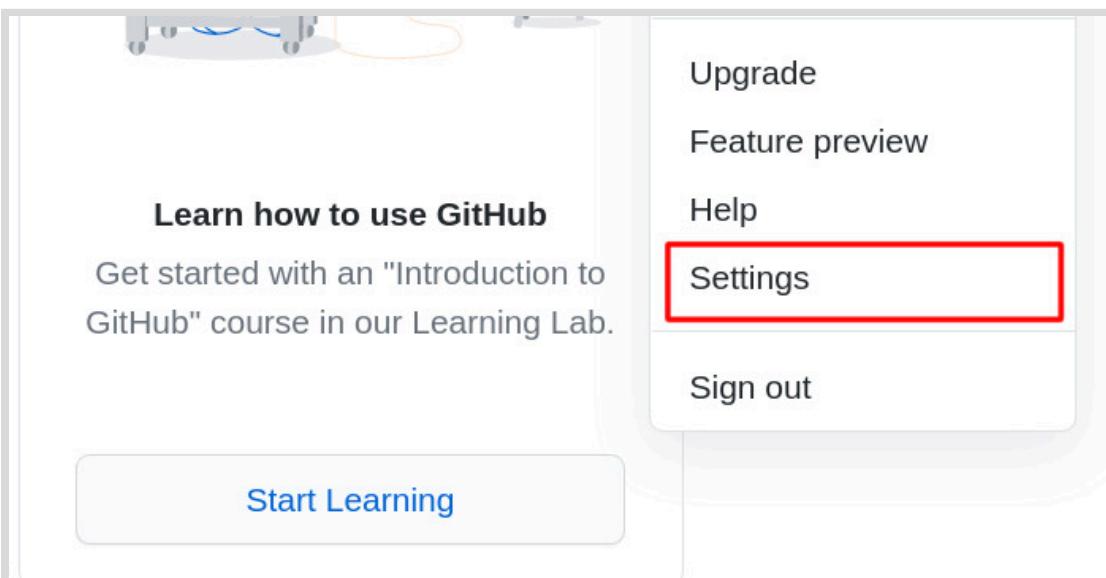


Abbildung 1.4: Menü „Settings“

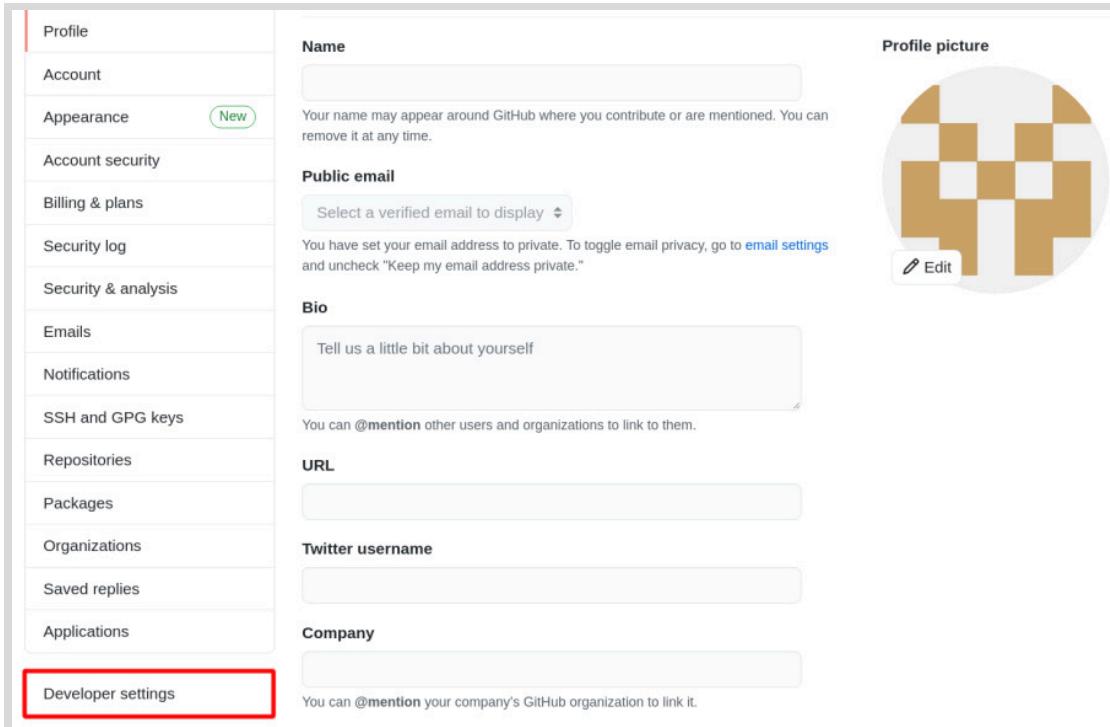


Abbildung 1.5: Developer settings

- 1.3. Wählen Sie im linken Bereich den Abschnitt mit dem persönlichen Zugriffstoken aus. Erstellen Sie auf der nächsten Seite Ihr neues Token, indem Sie auf **Generate new token** klicken. Sie werden dann aufgefordert, Ihr Passwort einzugeben.

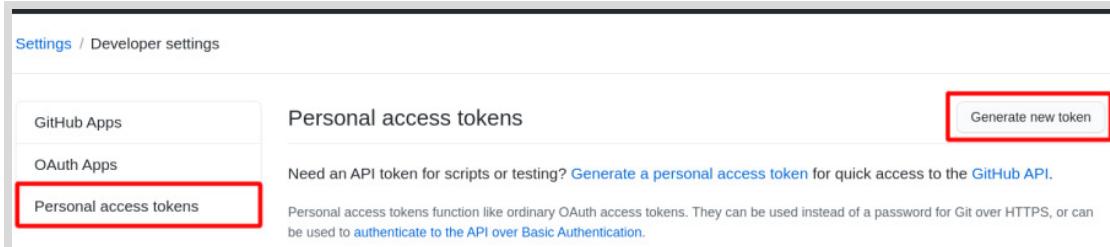


Abbildung 1.6: Bereich für persönliches Zugriffstoken

- 1.4. Geben Sie in das Feld **Note** eine kurze Beschreibung Ihres neuen Zugriffstokens ein.
- 1.5. Wählen Sie die Option **public_repo** aus, und lassen Sie die anderen Optionen deaktiviert. Erstellen Sie Ihr neues Zugriffstoken, indem Sie auf **Generate token** klicken.

New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note

Course DO280

What's this token for?

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

- | | |
|---|--------------------------------------|
| <input type="checkbox"/> repo | Full control of private repositories |
| <input type="checkbox"/> repo:status | Access commit status |
| <input type="checkbox"/> repo_deployment | Access deployment status |
| <input checked="" type="checkbox"/> public_repo | Access public repositories |
| <input type="checkbox"/> repo:invite | Access repository invitations |
| <input type="checkbox"/> security_events | Read and write security events |

Abbildung 1.7: Konfiguration des persönlichen Zugriffstokens

- 1.6. Ihr neues persönliches Zugriffstoken wird in der Ausgabe angezeigt. Erstellen Sie mit Ihrem bevorzugten Texteditor eine neue Datei im Benutzerverzeichnis des Kursteilnehmers mit dem Namen `token`, und fügen Sie Ihr generiertes persönliches Zugriffstoken ein.



Wichtig

Das persönliche Zugriffstoken kann nicht erneut in GitHub angezeigt werden. Es ist wichtig, das Token an einem sicheren Ort zu speichern. Falls Sie das persönliche Zugriffstoken erneut benötigen, müssen Sie ein neues erstellen.

Personal access tokens

[Generate new token](#)

[Revoke all](#)

Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your new personal access token now. You won't be able to see it again!

✓ ghp_kgYGzWcGE1CrdovkzuzeLTwvYY6eBX2l0vCK

[Delete](#)

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Abbildung 1.8: Generiertes Zugriffstoken



Wichtig

Wenn Sie in diesem Kurs bei der Verwendung von Git-Vorgängen in der Befehlszeile zur Eingabe eines Passworts aufgefordert werden, verwenden Sie Ihr Zugriffstoken als Passwort.

- 2. Sie müssen Ihre `workstation`-VM konfigurieren.

Verwenden Sie für die folgenden Schritte die Werte, die von der Red Hat Training Online Learning-Umgebung bei der Einrichtung Ihrer Online-Lab-Umgebung bereitgestellt werden:

The screenshot shows the 'Lab Environment' tab selected in the navigation bar. A red box highlights the 'OpenShift Details' section, which contains the following configuration parameters:

Username	RHT_OCP4_DEV_USER	youruser
Password	RHT_OCP4_DEV_PASSWORD	yourpassword
API Endpoint	RHT_OCP4_MASTER_API	https://api.cluster.domain.example.com:6443
Console Web Application		https://console-openshift-console.apps.cluster.domain.example.com
Cluster Id		your-cluster-id

Below this, there is a table showing the status of two environments: 'workstation' and 'classroom'. Both are listed as 'active'. To the right of each row are 'ACTION' and 'OPEN CONSOLE' buttons.

Öffnen Sie ein Terminal auf Ihrer `workstation`-VM und führen Sie den folgenden Befehl aus. Beantworten Sie die interaktiven Eingabeaufforderungen, um Ihre Workstation zu konfigurieren, bevor Sie eine andere Übung in diesem Kurs beginnen.

Wenn Sie einen Fehler machen, können Sie den Befehl jederzeit mit `Strg+C` unterbrechen und von vorne beginnen.

```
[student@workstation ~]$ lab-configure
```

- 2.1. Der Befehl `lab-configure` startet mit der Anzeige einer Reihe interaktiver Eingabeaufforderungen und versucht, sinnvolle Standardwerte für einige davon zu finden.

This script configures the connection parameters to access the OpenShift cluster for your lab scripts

- Enter the API Endpoint: <https://api.cluster.domain.example.com:6443> ①
- Enter the Username: `youruser` ②
- Enter the Password: `yourpassword` ③

```
• Enter the GitHub Account Name: yourgituser ④  
• Enter the Quay.io Account Name: yourquayuser ⑤  
  
...output omitted...
```

- ① Die URL zur Master-API Ihres OpenShift-Clusters. Geben Sie die URL als eine einzelne Zeile ohne Leerzeichen oder Zeilenumbrüche ein. Red Hat Training stellt diese Informationen bereit, wenn Sie Ihre Lab-Umgebung einrichten. Sie benötigen diese Informationen, um sich beim Cluster anzumelden sowie um Container-Anwendungen bereitzustellen.
- ② ③ Ihr OpenShift-Entwicklerbenutzername und das zugehörige Passwort. Red Hat Training stellt diese Informationen bereit, wenn Sie Ihre Lab-Umgebung einrichten. Sie müssen diesen Benutzernamen und das Passwort verwenden, um sich bei OpenShift anzumelden. Sie verwenden Ihren Benutzernamen auch als Teil von Identifikatoren wie Routen-Hostnamen und Projektnamen, um Kollisionen mit Identifikatoren anderer Kursteilnehmer zu vermeiden, die denselben OpenShift-Cluster wie Sie nutzen.
- ④ ⑤ Die Namen Ihrer persönlichen GitHub- und Quay.io-Benutzerkonten. Sie benötigen gültige, kostenlose Benutzerkonten für diese Online-Services, um die Übungen dieses Kurses durchzuführen. Wenn Sie noch keinen dieser Online-Services verwendet haben, finden Sie in *Anhang A, Erstellen eines GitHub-Benutzerkontos* und *Anhang B, Erstellen eines Quay-Benutzerkontos* Informationen zur Registrierung.

- 2.2. Mit dem Befehl `lab-configure` werden alle von Ihnen eingegebenen Informationen ausgegeben und es wird versucht, eine Verbindung zu Ihrem OpenShift-Cluster herzustellen:

```
...output omitted...
```

You entered:

```
• API Endpoint: https://api.cluster.domain.example.com:6443  
• Username: youruser  
• Password: yourpassword  
• GitHub Account Name: yourgituser  
• Quay.io Account Name: yourquayuser
```

```
...output omitted...
```

- 2.3. Wenn `lab-configure` ein Problem findet, wird eine Fehlermeldung angezeigt und der Befehl beendet. Sie müssen Ihre Angaben überprüfen und den Befehl `lab-configure` erneut ausführen. Die folgende Auflistung zeigt ein Beispiel für einen Verifizierungsfehler:

```
...output omitted...
```

```
Verifying your Master API URL...
```

ERROR:

Cannot connect to an OpenShift 4.5 API using your URL.

Please verify your network connectivity and that the URL does not point to an OpenShift 3.x nor to a non-OpenShift Kubernetes API.

No changes made to your lab configuration.

- 2.4. Wenn soweit alles in Ordnung ist, versucht `lab-configure` auf Ihre öffentlichen GitHub- und Quay.io-Benutzerkonten zuzugreifen:

```
...output omitted...
```

```
Verifying your GitHub account name...
```

```
Verifying your Quay.io account name...
```

```
...output omitted...
```

- 2.5. Wenn Probleme gefunden werden, zeigt `lab-configure` erneut eine Fehlermeldung an und wird beendet. Sie müssen Ihre Angaben überprüfen und den Befehl `lab-configure` erneut ausführen. Die folgende Auflistung zeigt ein Beispiel für einen Verifizierungsfehler:

```
...output omitted...
```

```
Verifying your GitHub account name...
```

ERROR:

Cannot find a GitHub account named: invalidusername.

No changes made to your lab configuration.

- 2.6. Schließlich überprüft der Befehl `lab-configure`, ob Ihr OpenShift-Cluster die erwartete Platzhalter-Domain ausgibt.

```
...output omitted...
```

```
Verifying your cluster configuration...
```

```
...output omitted...
```

- 2.7. Wenn alle Prüfungen erfolgreich sind, speichert der Befehl `lab-configure` Ihre Konfiguration:

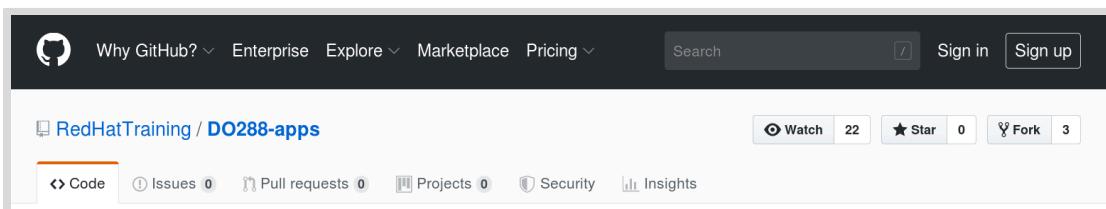
```
...output omitted...

Saving your lab configuration file...

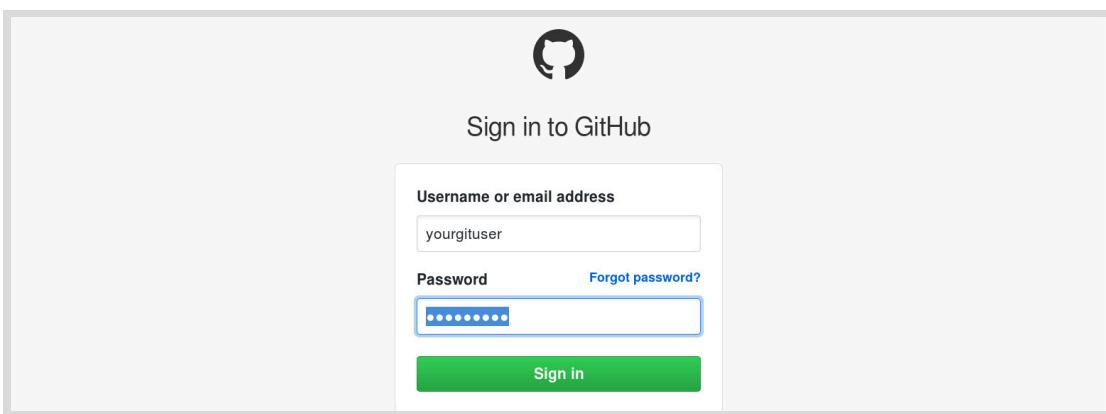
Saving your Maven settings file...

All fine, lab config saved. You can now proceed with your exercises.
```

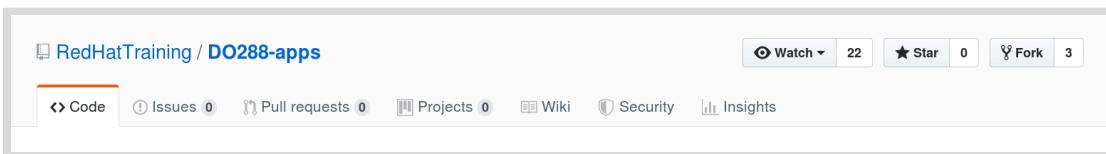
- 2.8. Wenn keine Fehler beim Speichern Ihrer Konfiguration vorliegen, sind Sie fast bereit, eine der Übungen dieses Kurses zu beginnen. Falls Fehler auftreten, versuchen Sie nicht, eine Übung zu beginnen, bevor Sie den Befehl `lab-configure` erfolgreich ausführen können.
- 3. Bevor Sie mit einer Übung beginnen, müssen Sie die Beispielanwendungen dieses Kurses in Ihr persönliches GitHub-Benutzerkonto forken. Führen Sie die folgenden Schritte aus:
- 3.1. Öffnen Sie einen Webbrowser und navigieren Sie zu <https://github.com/RedHatTraining/D0288-apps>. Wenn Sie nicht bei GitHub angemeldet sind, klicken Sie in der rechten oberen Ecke auf **Sign in**.



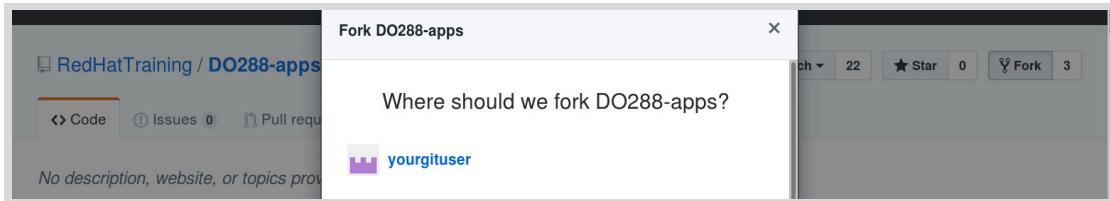
- 3.2. Melden Sie sich bei GitHub mit Ihrem persönlichen Benutzernamen und dem Passwort an.



- 3.3. Kehren Sie zum Repository RedHatTraining/D0288-apps zurück und klicken Sie in der rechten oberen Ecke auf **Fork**.

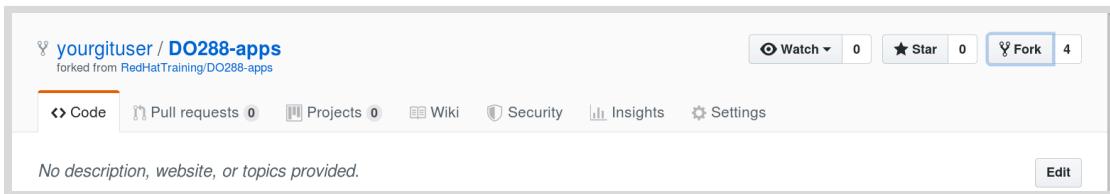


- 3.4. Klicken Sie im Fenster **Fork D0288-apps** auf **yourgituser**, um Ihr persönliches GitHub-Projekt auszuwählen.

**Wichtig**

Es ist zwar möglich, Ihren persönlichen Fork des <https://github.com/RedHatTraining/DO288-apps>-Repository umzubenennen, aber für die Auswertungsskripts, Hilfsskripts und die Beispielausgabe in diesem Kurs ist es erforderlich, dass Sie beim Forken des Repository den Namen **D0288-apps** beibehalten.

- 3.5. Nach ein paar Minuten wird in der GitHub-Weboberfläche Ihr neues Repository **yourgituser/DO288-apps** angezeigt.



- 4. Bevor Sie mit einer Übung beginnen, müssen Sie die Beispielanwendungen dieses Kurses aus Ihrem persönlichen GitHub-Benutzerkonto auf Ihre **workstation**-VM klonen. Führen Sie die folgenden Schritte aus:

- 4.1. Führen Sie den folgenden Befehl aus, um das Beispielanwendungs-Repository des Kurses zu klonen. Ersetzen Sie **yourgituser** durch den Namen Ihres persönlichen GitHub-Benutzerkontos:

```
[student@workstation ~]$ git clone https://github.com/yourgituser/D0288-apps
Cloning into 'D0288-apps'
...output omitted...
```

- 4.2. Überprüfen Sie, ob es sich bei **/home/student/D0288-apps** um ein Git-Repository handelt.

```
[student@workstation ~]$ cd D0288-apps
[student@workstation D0288-apps]$ git status
# On branch main

nothing to commit, working directory clean
```

- 4.3. Überprüfen Sie, ob **/home/student/D0288-apps** die Beispielanwendungen dieses Kurses enthält, und wechseln Sie zurück in den Benutzerordner des Benutzers **student**.

```
[student@workstation D0288-apps]$ head README.md
# D0288 Containerized Example Applications
...output omitted...
[student@workstation D0288-apps]$ cd ~
[student@workstation ~]$
```

- 5. Nachdem Sie nun einen lokalen Klon des D0288-apps-Repository auf Ihrer workstation-VM erstellt und den Befehl `lab-configure` erfolgreich ausgeführt haben, können Sie die Übungen dieses Kurses beginnen.

In diesem Kurs starten alle Übungen, die Anwendungen aus Quellcode erstellen, aus dem `main`-Branch des D0288-apps-Git-Repository. Bei Übungen, die Änderungen am Quellcode vornehmen, müssen Sie neue Branches erstellen, um Ihre Änderungen zu hosten, sodass der `main`-Branch immer einen als fehlerfrei bekannten Startpunkt enthält. Wenn Sie aus irgendeinem Grund eine Übung unterbrechen oder neu starten müssen und die Änderungen, die Sie in Ihren Git-Banches vornehmen, entweder speichern oder verwerfen müssen, finden Sie weitere Informationen unter *Anhang C, Nützliche Git-Befehle*.

- 6. Hiermit ist die angeleitete Übung beendet.

Bereitstellen einer Anwendung auf einem OpenShift-Cluster

Ziele

Nach Abschluss dieses Abschnitts sollten Sie zu Folgendem in der Lage sein:

- Bereitstellen einer Anwendung auf einem Cluster aus einem Dockerfile mit der CLI
- Beschreiben von Ressourcen, die mit dem Befehl `oc new-app` und der Web Console in einem Projekt erstellt werden

Entwicklungswege

Red Hat OpenShift Container Platform wurde für die Erstellung und Bereitstellung containerisierter Anwendungen konzipiert. OpenShift unterstützt zwei Hauptanwendungsfälle:

- Der gesamte Lifecycle der Anwendung von der Entwicklung bis zur Produktion wird mithilfe von OpenShift-Tools verwaltet.
- Vorhandene containerisierte Anwendungen, die außerhalb von OpenShift erstellt werden, werden in OpenShift bereitgestellt.



Wichtig

In OpenShift 4.6 erzeugt der Befehl `oc new-app` nun standardmäßig Deployment-Ressourcen anstatt DeploymentConfig-Ressourcen. Diese Version von DO288 verwendet DeploymentConfigs nur dann, wenn es erforderlich ist. Um DeploymentConfig-Ressourcen zu erstellen, können Sie das Flag `--as-deployment-config` beim Aufrufen von `oc new-app` übergeben. Weitere Informationen finden Sie unter Informationen zu Deployments und DeploymentConfigs [<https://docs.openshift.com/container-platform/4.6/applications/deployments/what-deployments-are.html>].

Mit dem Befehl `oc new-app` werden die zur Erstellung und Bereitstellung einer Anwendung in OpenShift erforderlichen Ressourcen erstellt. Es werden, dem jeweiligen Anwendungsfall entsprechend, unterschiedliche Ressourcen erstellt:

- Wenn der gesamte Lifecycle einer Anwendung von OpenShift verwaltet werden soll, erstellt der Befehl `oc new-app` eine Build-Konfiguration zur Verwaltung des Build-Prozesses, über den das Container-Image der Anwendung erstellt wird. Der Befehl `oc new-app` erstellt außerdem eine Deployment-Ressource zur Verwaltung des Deploymentprozesses, mit der das erzeugte Container-Image im OpenShift-Cluster ausgeführt wird. Im folgenden Beispiel delegieren Sie den gesamten Lifecycle an den OpenShift-Cluster: das Klonen eines Git-Repository, das Erstellen eines Container-Images und die Bereitstellung in einem OpenShift-Cluster.

```
[user@host ~]$ oc new-app \
https://github.com/RedHatTraining/DO288/tree/main/apps/apache-httppd
```

Wenn die URL auf ein Git-Repository verweist, können Sie optional einen Branch-Namen mithilfe des Raute-Trennzeichens (#) angeben. Die folgende URL gibt beispielsweise an, dass der Branch

my-branch verwendet werden soll: <https://github.com/RedHatTraining/D0288-apps#my-branch>.

- Wenn Sie eine vorhandene, containerisierte Anwendung in OpenShift bereitstellen möchten, erstellen Sie mit dem Befehl `oc new-app` eine Deployment-Ressource zur Verwaltung des Deploymentprozesses, mit der das vorhandene Container-Image im OpenShift-Cluster ausgeführt wird. Im folgenden Beispiel verweisen Sie mit der Option `--docker-image` auf ein Container-Image:

```
[user@host ~]$ oc new-app --docker-image=registry.access.redhat.com/rhel7-mysql57
```

Der Befehl `oc new-app` erstellt zudem einige zusätzliche Ressourcen wie Services und Image-Streams. Diese Ressourcen sind für die Verwaltung containerisierter Anwendungen mit OpenShift erforderlich und werden nachfolgend in diesem Kurs erläutert.

Mit der Schaltfläche **Add to Project** in der Web Console werden dieselben Aufgaben wie bei Verwendung des Befehls `oc new-app` ausgeführt. In einem späteren Kapitel dieses Kurses werden die OpenShift Web Console und ihre Verwendung behandelt.

Beschreiben der `oc new-app`-Befehlsoptionen

Der Befehl `oc new-app` übernimmt in seiner einfachsten Form ein einziges URL-Argument, das entweder auf ein Git-Repository oder ein Container-Image verweist. Der Befehl greift auf die URL zu, um zu ermitteln, wie das Argument zu interpretieren und ein Build oder eine Bereitstellung durchzuführen ist.

Der Befehl `oc new-app` erzielt möglicherweise nicht das von Ihnen gewünschte Ergebnis. Beispiel:

- Wenn ein Git-Repository sowohl ein Dockerfile als auch eine `index.php`-Datei enthält, kann OpenShift den zu verfolgenden Ansatz nur dann ermitteln, wenn dies explizit erwähnt wird.
- Wenn ein Git-Repository PHP-Quellcode enthält, aber der OpenShift-Cluster die Bereitstellung von PHP-Version 5.6 oder 7.0 unterstützt, schlägt der Build-Prozess fehl, da nicht klar ist, welche Version verwendet werden soll.

Um diese und andere Szenarien zu berücksichtigen, bietet der Befehl `oc new-app` eine Reihe von Optionen, mit denen Sie genau festlegen können, wie die Anwendung erstellt werden soll:

Unterstützte Optionen

Option	Beschreibung
<code>--as-deployment-config</code>	Konfiguriert <code>oc new-app</code> so, dass eine DeploymentConfig-Ressource anstatt einer Deployment-Ressource erstellt wird.
<code>--image-stream -i</code>	Stellt den Image-Stream bereit, der als S2I-Builder-Image für einen S2I-Build oder zur Bereitstellung eines Container-Images verwendet wird.
<code>--strategy</code>	<code>docker</code> oder <code>pipeline</code> oder <code>source</code>
<code>--code</code>	Stellt die URL zu einem Git-Repository bereit und wird als Eingabe für einen S2I-Build verwendet.

Option	Beschreibung
--docker-image	Liefert die URL zu einem Container-Image, das bereitgestellt werden soll.
--dry-run	Ist auf true festgelegt, um das Ergebnis des Vorgangs ohne Ausführung zu zeigen.
--context-dir	Gibt den Pfad zu dem Verzeichnis an, das das Stammverzeichnis (root) sein soll.

Verwalten des gesamten Application Lifecycle mit OpenShift

OpenShift verwaltet einen Application Lifecycle mit dem Prozess *Source-to-Image (S2I)*. S2I verwendet den Anwendungsquellcode aus einem Git-Repository, kombiniert diesen mit einem Basis-Container-Image, erstellt die Quelle und anschließend ein Container-Image mit einer startbereiten Anwendung.

Der Befehl `oc new-app` verwendet eine Git-Repository-URL als Eingabeargument und überprüft den Anwendungsquellcode, um zu bestimmen, welches Builder-Image zur Erstellung des Anwendungscontainer-Images verwendet werden soll:

```
[user@host ~]$ oc new-app http://gitserver.example.com/mygitrepo
```

Der Befehl `oc new-app` kann optional den Namen des Builder-Image-Streams als Argument übernehmen, entweder als Teil der Git-URL mit vorangestellter Tilde (~) oder als `--image-stream`-Argument (Kurzform: -i).

Die folgenden zwei Befehle zeigen die Verwendung eines PHP-S2I-Builder-Images:

```
[user@host ~]$ oc new-app php-http://gitserver.example.com/mygitrepo
```

```
[user@host ~]$ oc new-app -i php http://gitserver.example.com/mygitrepo
```

Dem Namen des Image-Streams kann optional ein bestimmtes Tag folgen, das in der Regel die Versionsnummer der Laufzeit für die Programmiersprache darstellt. Beispiel:

```
[user@host ~]$ oc new-app php:7.0~http://gitserver.example.com/mygitrepo
```

```
[user@host ~]$ oc new-app -i php:7.0 http://gitserver.example.com/mygitrepo
```

Angeben des Image-Stream-Namens

Einige Entwickler bevorzugen die Option -i anstelle der Tilde-Notation, da die Tilde je nach Bildschirmschriftart möglicherweise nicht gut lesbar ist. Die folgenden drei Befehle liefern dieselben Ergebnisse:

```
[user@host ~]$ oc new-app \
myis~http://gitserver.example.com/mygitrepo
```

```
[user@host ~]$ oc new-app \
-i myis http://gitserver.example.com/mygitrepo
```

```
[user@host ~]$ oc new-app -i myis --strategy source \
--code http://gitserver.example.com/mygitrepo
```

Obwohl der Befehl `oc new-app` eine bequeme Möglichkeit zur Bereitstellung von Anwendungen sein soll, müssen sich Entwickler darüber im Klaren sein, dass der Befehl versucht, die Ausgangssprache des angegebenen Git-Repository zu „erraten“.

Wenn im vorherigen Beispiel `myis` kein standardmäßiger S2I-Image-Stream ist, der von OpenShift bereitgestellt wird, liefert nur das erste Beispiel das gewünschte Ergebnis. Die Tilde-Notation deaktiviert die Spracherkennungsfunktion des Befehls `oc new-app`. Sie ermöglicht die Verwendung eines Image-Streams, der auf einen Builder für eine Programmiersprache verweist, die dem Befehl `oc new-app` nicht bekannt ist.

Die Tilde (~) und die `--image-stream (-i)`-Optionen funktionieren nicht auf dieselbe Weise. Die Option `-i` erfordert, dass der Git-Client lokal installiert ist, da die Spracherkennung das Repository klonen muss, damit es das Projekt überprüfen kann. Für die Tilde-Notation (~) ist dies nicht erforderlich.

Bereitstellen vorhandener, containerisierter Anwendungen für OpenShift

Wenn Sie außerhalb von OpenShift eine Anwendung entwickeln und das Anwendungscontainer-Image in einer Container-Image-Registry verfügbar ist, auf die über den OpenShift-Cluster zugegriffen werden kann, dann kann der Befehl `oc new-app` die Container-Image-URL als Eingabeargument übernehmen:

```
[user@host ~]$ oc new-app \
registry.example.com/mycontainerimage
```

Beachten Sie, dass bei Verwendung des vorherigen Befehls nicht bestimmt werden kann, ob die URL auf ein Git-Repository oder auf ein Container-Image in einer Registry verweist. Mit dem Befehl `oc new-app` wird auf die Eingabe-URL zugegriffen, um diese Mehrdeutigkeit aufzulösen. OpenShift untersucht den Inhalt der URL und bestimmt, ob es sich um einen Quellcode oder eine Container-Image-Registry handelt. Verwenden Sie entweder die Option `--code` oder `--docker-image`, um diese Mehrdeutigkeit zu vermeiden. Beispiel:

```
[user@host ~]$ *oc new-app \
--code http://gitserver.example.com/mygitrepo
```

```
[user@host ~]$ *oc new-app \
--docker-image registry.example.com/mycontainerimage
```

Bereitstellen vorhandener Dockerfiles mit OpenShift

Oftmals haben Sie vorhandene Container-Images, die mit Dockerfiles erstellt wurden. Wenn über ein Git-Repository auf die Dockerfiles zugegriffen werden kann, können Sie mit dem Befehl `oc new-app` eine Build-Konfiguration erstellen, mit der das Dockerfile-Build im OpenShift-Cluster

Kapitel 1 | Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster

durchgeführt wird. Anschließend wird das resultierende Container-Image an die interne Registry übertragen:

```
[user@host ~]$ oc new-app \
http://gitserver.example.com/mydockerfileproject
```

OpenShift greift auf die Quell-URL zu, um zu ermitteln, ob ein Dockerfile enthalten ist. Wenn dasselbe Projekt Quelldateien für Programmiersprachen enthält, kann OpenShift unter Umständen anstelle eines Dockerfile-Builds eine Builder-Konfiguration für einen S2I-Build erstellen. Verwenden Sie zur Vermeidung dieser Mehrdeutigkeit die Option `--strategy`:

```
[user@host ~]$ oc new-app \
--strategy docker http://gitserver.example.com/mydockerfileproject
```

Das folgende Beispiel veranschaulicht die Verwendung der Option `--strategy` für einen S2I-Build:

```
[user@host ~]$ oc new-app \
--strategy source http://gitserver.example.com/user/mygitrepo
```

Andere Optionen wie `--image-stream` und `--code` können zusammen mit der Option `--strategy` im selben Befehl verwendet werden.



Anmerkung

Der Befehl `oc new-app` bietet einige Optionen zum Erstellen von Anwendungen aus einem Template oder Anwendungen, die anhand einer Jenkins-Pipeline erstellt wurden. Diese Optionen werden im aktuellen Kapitel jedoch nicht behandelt.

Ressourcen, die mit dem Befehl `oc new-app` erstellt wurden

Der Befehl `oc new-app` fügt dem aktuellen Projekt die folgenden Ressourcen hinzu, um die Erstellung und Bereitstellung einer Anwendung zu unterstützen:

- Eine Build-Konfiguration zur Erstellung des Anwendungscontainer-Images aus Quellcode oder einem Dockerfile.
- Einen Image-Stream, der entweder auf das in der internen Registry erstellte Image oder auf ein vorhandenes Image in einer externen Registry verweist.
- Eine Deployment-Ressource, bei der der Image-Stream als Eingabe zur Erstellung von Anwendungs-Pods verwendet wird.
- Einen Service für alle Ports, die vom Anwendungscontainer-Image bereitgestellt werden. Wenn das Anwendungscontainer-Image keine Ports deklariert, dann erstellt der Befehl `oc new-app` keinen Service.

Diese Ressourcen initiieren eine Reihe von Prozessen, die wiederum weitere Ressourcen im Projekt erstellen, zum Beispiel die Anwendungs-Pods zur Ausführung containerisierter Anwendungen.

Der folgende Befehl erstellt eine Anwendung basierend auf dem `mysql`-Image mit der Bezeichnung `db=mysql`:

```
[user@host ~]$ oc new-app \
mysql -e MYSQL_USER=user -e MYSQL_PASSWORD=pass \
-e MYSQL_DATABASE=testdb -l db=mysql
```

In der folgenden Abbildung werden die Kubernetes- und OpenShift-Ressourcen angezeigt, die durch den Befehl `oc new-app` erstellt werden, wenn das Argument ein Container-Image ist:

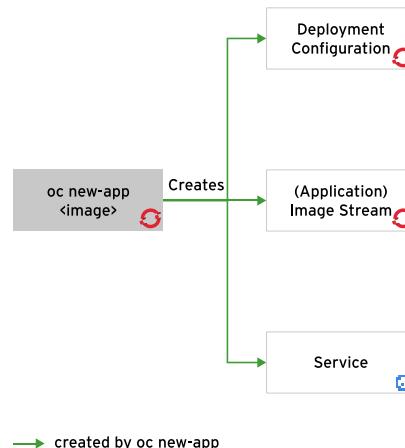


Abbildung 1.15: Ressourcen, die mit dem Befehl `oc new-app` erstellt wurden

Mit dem folgenden Befehl unter Verwendung einer Deploymentkonfiguration wird eine Anwendung aus dem Quellcode in der PHP-Programmiersprache erstellt:

```
[user@host ~]$ oc new-app --as-deployment-config \
--name hello -i php \
--code http://gitserver.example.com/mygitrepo
```

Führen Sie nach Abschluss des Builds und der Deploymentprozesse den Befehl `oc get all` aus, um alle Ressourcen im `test`-Projekt anzuzeigen. In der Ausgabe werden neben den Ressourcen, die vom Befehl `oc new-app` erstellt wurden, einige weitere Ressourcen angezeigt:

NAME	TYPE	FROM	LATEST			
bc/hello	Source	Git	3	①		
builds/hello-1	Source	Git@3a0af02	Complete	About an hour ago	1m16s	②
is/hello	DOCKER REPO	docker-registry.default.svc:5000/test/hello		TAGS	UPDATED	③
dc/hello	REVISION	DESIRED	CURRENT	TRIGGERED BY		④
rc/hello-1	1	1	1	config,image(hello:latest)		⑤
svc/hello	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE		⑥
	172.30.2.186	<none>	8080/TCP	2m31s		

Kapitel 1 | Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster

NAME	READY	STATUS	RESTARTS	AGE
po/hello-1-build	0/1	Completed	0	2m11s 7
po/hello-1_tmf1	1/1	Running	0	1m23s 8

- ① Die Build-Konfiguration, die mit dem Befehl `oc new-app` erstellt wurde.
- ② Der erste Build wird durch den Befehl `oc new-app` ausgelöst.
- ③ Der Image-Stream, der vom Befehl `oc new-app` erstellt wurde. Dieser verweist auf das Container-Image, das während des S2I-Prozesses erstellt wurde.
- ④ Die Deploymentkonfiguration, die mit dem Befehl `oc new-app` erstellt wurde.
- ⑤ Die Konfiguration für den Replication Controller, der bei der ersten Bereitstellung erstellt wurde. Bei nachfolgenden Bereitstellungen können außerdem Deployment-Pods erstellt werden.
- ⑥ Der Service, der infolge der Bereitstellung von Port 8080/TCP durch das PHP-S2I-Builder-Image vom Befehl `oc new-app` erstellt wurde.
- ⑦ Build-Pods aus den neuesten Builds werden zur Überprüfung dieser Logs von OpenShift beibehalten. Deployment-Pods werden nach der erfolgreichen Fertigstellung gelöscht.
- ⑧ Der Anwendungs-Pod, der bei der ersten Bereitstellung erstellt wurde.

Mit dem folgenden Befehl unter Verwendung einer Bereitstellung statt einer Deploymentkonfiguration wird eine Anwendung aus dem Quellcode in der PHP-Programmiersprache erstellt:

```
[user@host ~]$ oc new-app \
--name hello -i php \
--code http://gitserver.example.com/mygitrepo
```

Führen Sie nach Abschluss des Builds und der Deploymentprozesse den Befehl `oc get all` aus, um alle Ressourcen im test-Projekt anzuzeigen. In der Ausgabe werden neben den Ressourcen, die vom Befehl `oc new-app` erstellt wurden, einige weitere Ressourcen angezeigt:

NAME	READY	STATUS	RESTARTS	AGE
pod/hello-1-build	0/1	Completed	0	2m11s 1
pod/hello-57f548f776-q86pg	1/1	Running	0	1m23s 2

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/hello	1/1	1	1	62s 3

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/hello-57f548f776	1	1	1	3m 4
replicaset.apps/hello-875348fa8e	0	0	0	4m

NAME	TYPE	FROM	LATEST
buildconfig.build.openshift.io/hello	Source	Git	3 5

NAME	TYPE	FROM	STATUS	STARTED
build.build.openshift.io/hello-1	Source	Git@3a0af02	Complete	About an hour ago 6
DURATION				

NAME	TAGS	UPDATED	IMAGE REPOSITORY
imagestream.image.openshift.io/hello	hello	latest 46 seconds ago 7	docker-registry.default.svc:5000/test/

- ① Build-Pods aus den neuesten Builds werden zur Überprüfung dieser Logs von OpenShift beibehalten. Deployment-Pods werden nach der erfolgreichen Fertigstellung gelöscht.
- ② Der Anwendungs-Pod, der bei der ersten Bereitstellung erstellt wurde.
- ③ Die Deploymentkonfiguration, die mit dem Befehl `oc new-app` erstellt wurde.
- ④ Die Konfiguration für den Replication Controller, der bei der ersten Bereitstellung erstellt wurde. Bei nachfolgenden Bereitstellungen können außerdem Deployment-Pods erstellt werden.
- ⑤ Die Build-Konfiguration, die mit dem Befehl `oc new-app` erstellt wurde.
- ⑥ Der erste Build wird durch den Befehl `oc new-app` ausgelöst.
- ⑦ Der Image-Stream, der vom Befehl `oc new-app` erstellt wurde. Dieser verweist auf das Container-Image, das während des S2I-Prozesses erstellt wurde.

Eine Anwendung benötigt unter Umständen eine Reihe von Ressourcen, die mit dem Befehl `oc new-app` nicht erstellt werden können, zum Beispiel Routen, Secrets und Anforderungen für persistente Volumes. Diese Ressourcen können nach oder vor Verwendung des Befehls `oc new-app` mithilfe anderer `oc`-Befehle erstellt werden.

Alle Ressourcen, die mit dem Befehl `oc new-app` erstellt wurden, enthalten die `app-`Bezeichnung. Der Wert der `app`-Bezeichnung stimmt mit dem Kurznamen des Git-Repository der Anwendung oder mit einem vorhandenen Container-Image überein. Um einen anderen Wert für die `app`-Bezeichnung anzugeben, verwenden Sie die Option `--name`, zum Beispiel:

```
[user@host ~]$ oc new-app \
--name test http://gitserver.example.com/mygitrepo
```

Sie können Ressourcen, die mit dem Befehl `oc new-app` erstellt wurden, mithilfe eines einzelnen `oc delete`-Befehls in Kombination mit der `app`-Bezeichnung löschen, ohne das gesamte Projekt zu löschen oder andere Ressourcen zu beeinträchtigen, die möglicherweise im Projekt enthalten sind. Mit dem folgenden Befehl werden alle Ressourcen gelöscht, die vom vorherigen `oc new-app`-Befehl erstellt wurden:

```
[user@host ~]$ oc delete all -l app=test
```

Verwenden Sie das Argument der Option `--name`, um den Basisnamen für die Ressourcen anzugeben, die mit dem Befehl `oc new-app` erstellt wurden, beispielsweise Build-Konfigurationen und Services.

Der Befehl `oc new-app` kann innerhalb desselben OpenShift-Projekts mehrmals ausgeführt werden, um nacheinander Anwendungen mit mehreren Containern zu erstellen. Beispiel:

- Führen Sie den Befehl `oc new-app` mit der URL aus, die auf das Container-Image einer MongoDB-Datenbank verweist, um einen Datenbank-Pod und einen Service zu erstellen.

Kapitel 1 | Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster

- Führen Sie anhand des Service, der beim ersten Aufruf erstellt wurde, den Befehl `oc new-app` mit der URL des Git-Repository für eine Node.js-Anwendung aus, die Zugriff auf die Datenbank erfordert.

Sie können zu einem späteren Zeitpunkt alle Ressourcen, die von beiden Befehlen erstellt wurden, in eine Vorlagendatei exportieren.

Wenn Sie Ressourcendefinitionen überprüfen möchten, ohne die Ressourcen im aktuellen Projekt zu erstellen, verwenden Sie die Option `-o`:

```
[user@host ~]$ oc new-app \
-o json registry.example.com/mycontainerimage
```

Die Ressourcendefinitionen werden an die Standardausgabe gesendet und können an eine Datei umgeleitet werden. Die erstellte Datei kann anschließend angepasst oder in eine Vorlagendefinition eingefügt werden.



Anmerkung

OpenShift bietet eine Reihe vordefinierter Vorlagen für gängige Szenarien wie Datenbanken in Kombination mit einer Anwendung. Beispiel: Bei Verwendung der Vorlage `rails-postgresql` werden ein Container-Image einer PostgreSQL-Datenbank bereitgestellt und eine Ruby on Rails-Anwendung aus Quellcode erstellt.

Führen Sie den Befehl `oc new-app -h` aus, um eine vollständige Liste der vom Befehl `oc new-app` unterstützten Optionen und eine Liste der Beispiele anzuzeigen.

Verweisen auf Container-Images mit Image-Streams und -Tags

Die OpenShift-Community empfiehlt die Verwendung von Image Stream-Ressourcen, um auf die Container-Images zu verweisen, anstatt der direkten Verwendung von Container-Images. Image-Stream-Ressourcen verweisen auf ein Container-Image in der internen oder in einer externen Registry und speichern Metadaten wie verfügbare Tags und Prüfsummen von Image-Inhalten.

Wenn in einem Image-Stream Container-Image-Metadaten enthalten sind, kann OpenShift auf Grundlage dieser Daten Vorgänge ausführen, wie das Zwischenspeichern von Images, anstatt hierfür auf einen Registry-Server zurückgreifen zu müssen. Dies ermöglicht auch die Verwendung von Benachrichtigungs- bzw. Pooling-Strategien, um auf Aktualisierungen von Image-Inhalten zu reagieren.

Build-Konfigurationen und Deploymentkonfigurationen verwenden Image-Stream-Ereignisse, um Vorgänge – wie die nachfolgend aufgeführten – auszuführen:

- Auslösen eines neuen S2I-Builds aufgrund der Aktualisierung des Builder-Images
- Auslösen einer neuen Pod-Bereitstellung für eine Anwendung aufgrund der Aktualisierung des Anwendungscontainer-Images in einer externen Registry

Der einfachste Weg zur Erstellung eines Image-Streams ist die Verwendung des Befehls `oc import-image` in Kombination mit der Option `--confirm`. Im folgenden Beispiel wird ein Image-Stream mit dem Namen `myis` für das Container-Image `acme/awesome` erstellt, das aus der ungesicherten Registry unter `registry.acme.example.com` stammt:

```
[user@host ~]$ oc import-image myis --confirm \
--from registry.acme.example.com:5000/acme/awesome --insecure
```

Das openshift-Projekt stellt eine Reihe von Image-Streams für alle OpenShift-Cluster-Benutzer bereit. Sie können eigene Image-Streams im aktuellen Projekt sowohl mit dem Befehl `oc new-app` als auch mit OpenShift-Vorlagen erstellen.

Eine Image-Stream-Ressource kann mehrere *Image-Stream-Tags* definieren. Ein Image-Stream-Tag kann entweder auf ein anderes Container-Image-Tag oder auf den Namen eines anderen Container-Images verweisen. Das bedeutet, dass Sie einfachere, kürzere Namen für häufig genutzte Images wie S2I-Builder-Images und verschiedene Namen oder Registrys für Variationen desselben Images verwenden können. Beispiel: Der Image-Stream `ruby` aus dem `openshift`-Projekt definiert die folgenden Image-Stream-Tags:

- `ruby:2.5` verweist auf `rhel8/ruby-25` aus dem Red Hat Container Catalog.
- `ruby:2.6` verweist auf `rhel8/ruby-26` aus dem Red Hat Container Catalog.



Literaturhinweise

Weitere Informationen finden Sie im Kapitel *Developer CLI commands* der *CLI-Referenz* für Red Hat OpenShift Container Platform 4.6 unter
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/cli_tools/index#cli-developer-commands

► Angeleitete Übung

Bereitstellen einer Anwendung auf einem OpenShift-Cluster

In dieser Übung verwenden Sie OpenShift zum Erstellen und Bereitstellen einer Anwendung aus einem Dockerfile.

Ergebnisse

Sie sollten in der Lage sein, eine Anwendung mithilfe der Docker-Build-Strategie zu erstellen und alle Ressourcen aus der Anwendung zu löschen, ohne das Projekt zu löschen.

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen aktiven OpenShift-Cluster
- Auf das übergeordnete Image für die Beispielanwendung (ubi8/ubi)
- Auf die Beispielanwendung im Git-Repository (ubi-echo)

Führen Sie den folgenden Befehl auf der workstation-VM aus, um die Voraussetzungen zu überprüfen und die Lösungsdateien herunterzuladen:

```
[student@workstation ~]$ lab docker-build start
```

Anweisungen

► 1. Überprüfen Sie das Dockerfile für die Beispielanwendung.

- 1.1. Wechseln Sie zu Ihrem lokalen Klon des D0288-apps-Git-Repository und checken Sie den main-Branch des Kurs-Repository aus, um sicherzustellen, dass Sie diese Übung von einem als fehlerfrei bekannten Zustand aus starten:

```
[student@workstation ~]$ cd D0288-apps
[student@workstation D0288-apps]$ git checkout main
...output omitted...
```

- 1.2. Erstellen Sie einen neuen Branch, um alle Änderungen zu speichern, die Sie während dieser Übung vornehmen:

```
[student@workstation D0288-apps]$ git checkout -b docker-build
Switched to a new branch 'docker-build'
[student@workstation D0288-apps]$ git push -u origin docker-build
...output omitted...
* [new branch]      docker-build -> docker-build
Branch docker-build set up to track remote branch docker-build from origin.
```

1.3. Überprüfen Sie das Dockerfile für die Anwendung im Ordner ubi-echo:

```
[student@workstation D0288-apps]$ cat ubi-echo/Dockerfile
FROM registry.access.redhat.com/ubi8/ubi:8.0 ①
USER 1001 ②
CMD bash -c "while true; do echo test; sleep 5; done" ③
```

- ① Das übergeordnete Image ist das Universal Base Image (UBI) für Red Hat Enterprise Linux 8.0 aus dem Red Hat Container Catalog.
- ② Die Benutzer-ID, mit der dieses Container-Image ausgeführt wird. Ein Wert ungleich Null würde hier funktionieren. Nur, um es von Standard-Systembenutzern zu unterscheiden, wie z. B. apache, die sich in der Regel im untersten Bereich der UID-Werte befinden.
- ③ Die Anwendung führt eine Schleife aus, die alle fünf Sekunden das Echo „test“ ausgibt.

► 2. Erstellen Sie das Anwendungscontainer-Image mithilfe des OpenShift-Clusters.

2.1. Laden Sie die Konfiguration Ihrer Kursumgebung.

Führen Sie den folgenden Befehl aus, um die in der ersten angeleiteten Übung erstellten Umgebungsvariablen zu laden:

```
[student@workstation D0288-apps]$ source /usr/local/etc/ocp4.config
```

2.2. Melden Sie sich bei OpenShift mit Ihrem Entwicklerbenutzernamen an:

```
[student@workstation D0288-apps]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

2.3. Erstellen Sie ein neues Projekt für die Anwendung. Stellen Sie dem Namen des Projekts den Entwicklerbenutzernamen voran.

```
[student@workstation D0288-apps]$ oc new-project ${RHT_OCP4_DEV_USER}-docker-build
Now using project "youruser-docker-build" on server "https://
api.cluster.domain.example.com:6443".
```

2.4. Erstellen Sie eine neue Anwendung mit dem Namen „echo“ aus dem Dockerfile im Ordner ubi-echo. Verwenden Sie den Branch, den Sie in einem vorherigen Schritt erstellt haben. Dadurch wird neben anderen Ressourcen eine Build-Konfiguration erstellt.

```
[student@workstation D0288-apps]$ oc new-app --name echo \
https://github.com/${RHT_OCP4_GITHUB_USER}/D0288-apps#docker-build \
--context-dir ubi-echo
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "ubi" created
imagestream.image.openshift.io "echo" created
```

```
buildconfig.build.openshift.io "echo" created
deployment.apps.openshift.io "echo" created
--> Success
...output omitted...
```

Ignorieren Sie die Warnungen zum Basis-Image, das als root ausgeführt wird. Denken Sie daran, dass Ihr Dockerfile zu einem unprivilegierten Benutzer gewechselt hat.

2.5. Beobachten Sie die Build-Logs:

```
[student@workstation D0288-apps]$ oc logs -f bc/echo
Cloning "https://github.com/youruser/D0288-apps#docker-build" ...
Replaced Dockerfile FROM image registry.access.redhat.com/ubi8/ubi:8.0
Caching blobs under "/var/cache/blobs".

...output omitted...
Pulling image registry.access.redhat.com/ubi8/ubi@sha256:1a2a...75b5
...output omitted...
STEP 1: FROM registry.access.redhat.com/ubi8/ubi@sha256:1a2a...75b5 ①
STEP 2: USER 1001
STEP 3: CMD bash -c "while true; do echo test; sleep 5; done"
STEP 4: ENV "OPENSHIFT_BUILD_NAME"="echo-1" ... ②
STEP 5: LABEL "io.openshift.build.commit.author"=...
STEP 6: COMMIT temp.builder.openshift.io/youruser-docker-build/echo-1:... ③
...output omitted...
Pushing image image-registry.openshift-image-registry.svc:5000/youruser-docker-
build/echo:latest ... ④
Push successful
```

- ① Der Befehl `oc new-app` identifiziert das Git-Repository richtigerweise als ein Dockerfile-Projekt und der OpenShift-Build führt einen Dockerfile-Build durch.
- ② OpenShift hängt mit den Anweisungen `ENV` und `LABEL` Metadaten an das Anwendungscontainer-Image an.
- ③ OpenShift übergibt das Anwendungs-Image an die Container-Engine des Knotens.
- ④ OpenShift überträgt das Anwendungs-Image von der Container-Engine des Knotens zur internen Registry des Clusters.

► 3. Überprüfen Sie, ob die Anwendung in OpenShift funktioniert.

3.1. Warten Sie, bis das Anwendungscontainer-Image bereitgestellt ist. Wiederholen Sie den Befehl `oc status`, bis die Ausgabe eine erfolgreiche Bereitstellung zeigt:

```
[student@workstation D0288-apps]$ oc status
In project youruser-docker-build on server
  https://api.cluster.domain.example.com:6443

  deployment/echo deploys istag/echo:latest <-
    bc/echo docker builds https://github.com/youruser/D0288-apps#docker-build on
    istag/ubi:8.0
```

```
deployment #2 running for 20 minutes - 1 pod
deployment #1 deployed 21 minutes ago
...output omitted...
```

Die erste Bereitstellung ist für den Builder-Pod. Die zweite Bereitstellung ist Ihr ausgeführter Anwendungs-Pod.

- 3.2. Warten Sie, bis der Anwendungs-Pod bereit ist und ausgeführt wird. Wiederholen Sie den Befehl `oc get pod`, bis in der Ausgabe in etwa Folgendes angezeigt wird:

```
[student@workstation D0288-apps]$ oc get pod
NAME        READY   STATUS    RESTARTS   AGE
echo-1-build 0/1     Completed  0          1m
echo-555xx   1/1     Running   0          14s
```

- 3.3. Zeigen Sie die Anwendungs-Pod-Logs an, um zu überprüfen, ob das Anwendungscontainer-Image die erwartete Ausgabe unter OpenShift generiert. Verwenden Sie den Namen des Anwendungs-Pod aus dem vorherigen Schritt.

```
[student@workstation D0288-apps]$ oc logs echo-555xx | tail -n 3
test
test
test
```

- 4. Überprüfen Sie die Build- und die Deploymentkonfiguration, um zu ermitteln, wie sie mit dem Image-Stream zusammenhängen.

- 4.1. Überprüfen Sie die Build-Konfiguration:

```
[student@workstation D0288-apps]$ oc describe bc echo
Name:           echo
...output omitted...
Labels:         app=echo
...output omitted...
Strategy:      Docker
URL:            https://github.com/youruser/D0288-apps
Ref:            docker-build ①
ContextDir:    ubi-echo ②
From Image:    ImageStreamTag ubi:8.0 ③
Output to:     ImageStreamTag echo:latest ④
...output omitted...
```

- ① Builds werden aus dem `docker-build`-Branch im Git-Repository im Attribut `URL` gestartet.
- ② Builds übernehmen im Attribut `URL` nur den Ordner `ubi-echo` aus dem Git-Repository.
- ③ Builds übernehmen einen Image-Stream, der auf das übergeordnete Image aus dem Dockerfile verweist, sodass neue Builds durch Image-Änderungen ausgelöst werden können.
- ④ Builds generieren ein neues Container-Image und übertragen es über einen Image-Stream an die interne Registry.

4.2. Überprüfen Sie den Image-Stream:

```
[student@workstation DO288-apps]$ oc describe is echo
Name:           echo
...output omitted...
Labels:         app=echo
...output omitted...
Image Repository: image-registry.openshift-image-registry.svc:5000/youruser-
docker-build/echo①
...output omitted...
latest
no spec tag

* image-registry.openshift-image-registry.svc:5000/youruser-docker-build/
echo@sha256:5bbf...ef0b ②
...output omitted...
```

- ① Der Image-Stream zeigt mit dem Service-DNS-Namen auf die interne OpenShift-Registry.
- ② Ein SHA256-Hash identifiziert das neueste Image. Mit diesem Hash kann der Image-Stream ermitteln, ob das Image geändert wurde.

4.3. Überprüfen Sie die Bereitstellung:

```
[student@workstation DO288-apps]$ oc describe deployment echo
Name:           echo
...output omitted...
Labels:         app=echo
Annotations:   deployment.kubernetes.io/revision: 2
               image.openshift.io/triggers: ①
               [{"from":{"kind":"ImageStreamTag","name":"echo:latest"},
                 "fieldPath":"spec.template.spec.containers[?(.name==\"echo
                 \")].image"}]
...output omitted...
Pod Template:
...output omitted...
Containers:
echo:
  Image:      image-registry.openshift-image-registry.svc:5000/youruser-
docker-build/echo@sha256:5bbf...ef0b ②
...output omitted...
Deployment #1 (latest):
...output omitted...
```

- ① Die Bereitstellung weist einen Trigger im Image-Stream auf. Wenn der Image-Stream geändert wird, dann wird eine neue Bereitstellung durchgeführt.
- ② Die Pod-Vorlage in der Deploymentkonfiguration gibt den SHA256-Hash des Container-Images an, um Deploymentstrategien wie Rolling Upgrades zu unterstützen.

► 5. Ändern Sie die Anwendung.

- 5.1. Bearbeiten Sie die CMD-Anweisung im Dockerfile unter ~/D0288-apps/ubi-echo/**Dockerfile**, um einen Zähler anzuzeigen. Der endgültige Inhalt des Dockerfile sollte wie folgt lauten:

```
FROM registry.access.redhat.com/ubi8/ubi:8.0
USER 1001
CMD bash -c "while true; do (( i++ )); echo test \$i; sleep 5; done"
```

- 5.2. Committen und pushen Sie die Änderungen an den Git-Server.

```
[student@workstation D0288-apps]$ cd ubi-echo
[student@workstation ubi-echo]$ git commit -a -m 'Add a counter'
...output omitted...
[student@workstation ubi-echo]$ git push
...output omitted...
[student@workstation ubi-echo]$ cd ~
[student@workstation ~]$
```

- 6. Erstellen Sie die Anwendung neu und überprüfen Sie, ob OpenShift das neue Container-Image bereitstellt.

- 6.1. Starten Sie einen neuen OpenShift-Build:

```
[student@workstation ~]$ oc start-build echo
build.build.openshift.io/echo-2 started
```

- 6.2. Beobachten Sie die Logs des neuen Builds und warten Sie, bis der Build abgeschlossen ist:

```
[student@workstation ~]$ oc logs -f bc/echo
...output omitted...
Push successful
```

- 6.3. Überprüfen Sie, ob OpenShift nach Abschluss des Builds eine neue Bereitstellung beginnt.

```
[student@workstation ~]$ oc status
...output omitted...
dc/echo deploys istag/echo:latest <-
bc/echo docker builds https://github.com/youruser/D0288-apps#docker-build on
istag/ubi:8.0
deployment #3 running for 43 seconds - 1 pod
deployment #2 deployed 26 minutes ago
deployment #1 deployed 27 minutes ago
...output omitted...
```

- 6.4. Warten Sie, bis der neue Anwendungs-Pod bereit ist und ausgeführt wird:

```
[student@workstation ~]$ oc get pod
NAME        READY   STATUS    RESTARTS   AGE
echo-1-build 0/1     Completed  0          27m
echo-2-build 0/1     Completed  0          1m
echo-p1hg    1/1     Running   0          1m
```

- 6.5. Zeigen Sie die Anwendungs-Pod-Logs an, um zu zeigen, dass er das neue Container-Image ausführt. Verwenden Sie den Pod-Namen aus dem vorherigen Schritt:

```
[student@workstation ~]$ oc logs echo-p1hg | head -n 3
test 1
test 2
test 3
```

- 7. Vergleichen Sie den Status des Image-Streams vor und nach der Neuerstellung der Anwendung.

Überprüfen Sie den aktuellen Status des Image-Streams:

```
[student@workstation ~]$ oc describe is echo
Name:      echo
...output omitted...
Labels:    app=echo
...output omitted...
latest
no spec tag

* image-registry.openshift-image-registry.svc:5000/youruser-docker-build/
echo@sha256:025a...542f ①
2 minutes ago
image-registry.openshift-image-registry.svc:5000/youruser-docker-build/
echo@sha256:5bbf...ef0b ②
...output omitted...
```

- ① Dies ist das neue Image. Beachten Sie, dass sich sein SHA256-Hash von dem des alten Images unterscheidet.
② Dies ist das alte Image.

- 8. Löschen Sie alle Anwendungsressourcen.

- 8.1. Führen Sie den Befehl `oc delete` mit der Anwendungsbezeichnung aus, die vom Befehl `oc new-app` generiert wurde.

```
[student@workstation ~]$ oc delete all -l app=echo
deployment.apps "echo" deleted
buildconfig.build.openshift.io "echo" deleted
build.build.openshift.io "echo-1" deleted
build.build.openshift.io "echo-2" deleted
imagestream.image.openshift.io "echo" deleted
imagestream.image.openshift.io "ubi" deleted
```

- 8.2. Vergewissern Sie sich, dass keine Ressourcen mehr im Projekt vorhanden sind. Alle Ressourcen aus der einzigen Anwendung im Projekt sollten gelöscht sein. Wenn in der Ausgabe ein Pod mit dem Status `Terminating` aufgeführt ist, führen Sie den Befehl so lange aus, bis der Pod nicht mehr angezeigt wird.

```
[student@workstation ~]$ oc get all  
No resources found.
```

Beenden

Führen Sie auf der `workstation` den Befehl `lab docker-build finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken.

```
[student@workstation ~]$ lab docker-build finish
```

Hiermit ist die angeleitete Übung beendet.

Verwalten von Anwendungen mit der Web Console

Ziele

Nach Abschluss dieses Abschnitts sollten Sie in der Lage sein, die Web Console für Folgendes zu verwenden:

- Bereitstellen einer Anwendung aus einem binären Image und Verwalten ihrer Ressourcen
- Anzeigen von Pod- und Build-Logs
- Bearbeiten von Ressourcendefinitionen

Überblick über die OpenShift Web Console

Die OpenShift Web Console ist eine browserbasierte Benutzeroberfläche, die eine grafische Alternative für die meisten allgemeinen Aufgaben bietet, die zum Verwalten von OpenShift-Projekten und -Anwendungen erforderlich sind. Die Funktionalität der Web Console ist in erster Linie auf Entwickleraufgaben und den Workflow ausgerichtet. Die Web Console bietet keine umfassende Cluster-Verwaltungsfunktionalität. Für diese Funktionalität ist in der Regel die Verwendung des Befehls `oc` erforderlich.

Verwenden Sie zum Aufrufen der Web Console die OpenShift API URL. Bei Clustern mit einem einzelnen Control Plane-Knoten ist dies in der Regel eine HTTPS-URL zum öffentlichen Hostnamen dieses Knotens.

Auf der Homepage der Web Console wird eine Liste der Projekte angezeigt, auf die der aktuelle Benutzer zugreifen kann. Von der Homepage aus können Sie neue Projekte erstellen, vorhandene löschen und zur Projektübersichtsseite navigieren.

Name	Display Name	Status	Requester
PR your-project	No display name	Active	youruser

Abbildung 1.16: Auflistung von Projekten in der Web Console

Kapitel 1 | Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster

Vermutlich werden Sie den Großteil Ihrer Zeit für die Projektübersichtsseiten und die vielen Seiten der Projektressource aufwenden. Auf der Projektübersichtsseite werden zusammenfassende Informationen zu Anwendungen im Projekt und zum Status sämtlicher Anwendungs-Pods angezeigt. Auf der Projektübersichtsseite können Sie zu Ressourcendetailseiten navigieren und dem Projekt Anwendungen hinzufügen.

The screenshot shows the Red Hat OpenShift Web Console interface. On the left, a sidebar menu includes Home, Projects, Search, Explore, and Events. Under Projects, 'your-project' is selected. The main content area shows 'Project Details' for 'PR your-project' (Active). It features tabs for Overview, Details, YAML, Workloads, and Role Bindings. The Overview tab is active, displaying sections for Details (Name: your-project, Requester: youruser, Labels: No labels), Status (Active), Utilization (CPU usage over 1 hour), and Inventory (0 Deployments, 1 Deployment Config, 0 Stateful Sets, 3 Pods). To the right, an Activity log lists recent events from 15:19, such as 'Created container...' and 'Successfully pulled...'. A 'Recent Events' section shows a pause button.

Abbildung 1.17: Projektübersicht der Web Console

Anwendungen in OpenShift

Die OpenShift Web Console definiert eine Anwendung als einen Satz von Ressourcen, die denselben Wert für die Bezeichnung `app` aufweisen. Der Befehl `oc new-app` fügt diese Bezeichnung zu allen Anwendungsressourcen hinzu, die er erstellt. Der Bereich `+Add` bietet in der **Developer**-Perspektive Funktionen, die dem Befehl `oc new-app` ähneln, einschließlich des Hinzufügens der Bezeichnung `app` zu Ressourcen.

The screenshot shows the Red Hat OpenShift Web Console interface with the 'Developer' perspective selected. The sidebar includes +Add, Topology, Monitoring, Search, Builds, Helm, and Project. The main content area is titled 'Add' and says 'Select a way to create an application, component or service from one of the options.' It features two cards: 'From Git' (represented by a GitHub icon) and 'Container Image' (represented by a Docker image icon). Both cards have descriptive text below them.

Abbildung 1.18: Aktionen, die über den Bereich „+Add“ verfügbar sind

Kapitel 1 | Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster

Über den Bereich **Add** wird ein Assistent aufgerufen, mit dem Sie aus S2I-Builder-Images, Vorlagen und Container-Images auswählen können, um für den OpenShift-Cluster eine Anwendung im Rahmen eines bestimmten Projekts bereitzustellen. Der Assistent kategorisiert Images und Vorlagen im Katalog entsprechend den Bezeichnungen in den Container-Images und Annotationen in den Vorlagen und Image-Stream-Ressourcen.

Ressourcendetailseiten

Auf den meisten Ressourcendetailseiten werden alle Projektressourcen eines bestimmten Typs aufgelistet. Sie enthalten Links für das Löschen spezifischer Ressourcen und für den Zugriff auf die Detailseite der jeweiligen Ressource.

Die Detailseite für eine einzelne Ressource enthält auf mehreren Tabs benutzerdefinierte Statusinformationen für den jeweiligen Ressourcentyp. Beispiel:

- Auf einer Build-Detailseite werden der Verlauf, die Konfiguration, Umgebung und Logs für den jeweiligen Build angezeigt.
- Auf einer Deploymentdetailseite werden der Verlauf, die Konfiguration, Umgebung, Ereignisse und Logs für die jeweilige Bereitstellung angezeigt.
- Auf einer Servicedetailseite werden der Satz von Pods, für die der Service einen Lastenausgleich durchgeführt hat, und auch die (ggf. vorhandenen) Routen angezeigt, die auf den Service verweisen.
- Auf einer Pod-Detailseite werden der Status, die Konfiguration, Umgebung, Logs und Ereignisse für den jeweiligen Pod angezeigt. Hier kann zudem eine Terminalsitzung geöffnet werden, die eine Shell in einem Container über den Pod ausführt.

Auf der Ressourcendetailseite werden in der Regel alle der jeweiligen Ressource zugeordneten Bezeichnungen aufgeführt. OpenShift verwendet Bezeichnungen, um Beziehungen zwischen Ressourcen zu erfassen. Beispielsweise weisen alle durch eine Bereitstellung erstellten Pods die Bezeichnung `deployment` mit dem Namen der Bereitstellung auf.

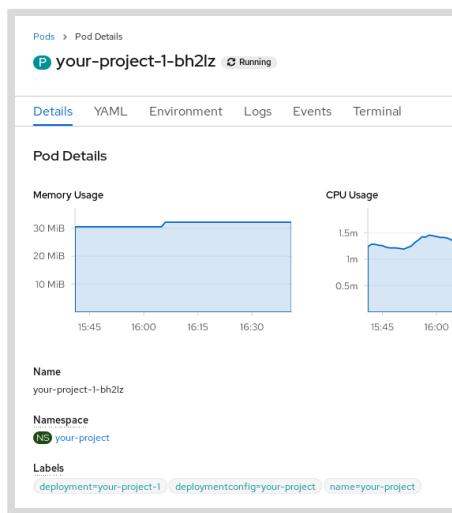


Abbildung 1.19: Bezeichnungen im unteren Bereich einer Pod-Detailseite

Wenn in der Web Console ein Ressourcename angezeigt wird, handelt es sich in der Regel um einen Link zur Detailseite der jeweiligen Ressource. Die Navigationsleiste auf der linken Seite der Web Console bietet Zugriff auf die Detailseite für alle unterstützten Ressourcentypen. Im oberen Bereich der Navigationsleiste bietet ein Startsymbol Zugriff auf die Seite mit der Projektliste.

Zugreifen auf Logs mit der Web Console

Die Pod-Detailseiten in der Web Console enthalten den Tab **Logs**, auf der die Pod-Logs angezeigt werden. Die Containerlaufzeit erfasst die Standardausgabe der Container in einem Pod und speichert sie als Pod-Logs.



Anmerkung

Nur Logs, die in die Standardausgabe im Container geschrieben werden, werden mit dem Befehl `oc logs` oder in der Web Console angezeigt. Wenn eine containerisierte Anwendung stattdessen ihre Logsreignisse in Dateien speichert, entweder im temporären Container-Storage oder auf einem persistenten Volume, zeigt OpenShift diese Logs weder in der Web Console noch durch Ausführen des Befehls `oc logs` an.

Auf dem Tab **Logs** werden die neuesten Logsinträge automatisch aktualisiert. Dieser Tab bietet auch die folgenden Aktionen:

- Mit dem Link **Download** laden Sie die Logs herunter und speichern sie in einer lokalen Datei.
- Verwenden Sie den Link **Expand**, damit die Pod-Logs den gesamten Bildschirm nutzen und die Anzeige einfacher lesbar ist.

Bei Builds und Bereitstellungen handelt es sich um Vorgänge, die von OpenShift mithilfe von Builder- und Deployment-Pods ausgeführt werden. Die Seite **Build Details** erfasst und speichert die Logs aus dem Builder-Pod. Verwenden Sie diese Seite, um diese Builder-Pod-Logs anzusehen. OpenShift speichert die Logs einer Bereitstellung nur, wenn während der Bereitstellung Fehler auftreten.

```

Builds > Build Details
B your-project-1 Complete Actions ▾

Details YAML Environment Logs Events

Log stream ended. Download | Expand

56 lines
Copying config sha256:c8fb8cb622514af059a359153cbf76b7db3270816d4b7ef5a48b2a0080f1dbb0
Writing manifest to image destination
Storing signatures
-> c8fb8cb622514af059a359153cbf76b7db3270816d4b7ef5a48b2a0080f1dbb0

Pushing image image-registry.openshift-image-registry.svc:5000/your-project/your-project:latest ...
Getting image source signatures
Copying blob sha256:02cb2f8840f27e51a3c232637b7496e15a9a571885a64f0d5e49dc2c1674fd
Copying blob sha256:9e7ad6c796f0a75c560158a9f9e30fb8b5a90cb53edce9ffbd5778406e4de39
Copying blob sha256:fcb206e9329a1674dd9e8efbee45c9be28dd0d9cbabba3c6bb67a2f22fcfcf2a
Copying blob sha256:e7021e0589e97471d99c4265b7c8e64da328e48f116b5f260353b2e02adb373
Copying blob sha256:9aa629e11c896fb5c92c3ede513fa50843042f978963a2a9e31288d1a7d5489
Copying config sha256:c8fb8cb622514af059a359153cbf76b7db3270816d4b7ef5a48b2a0080f1dbb0
Writing manifest to image destination
Storing signatures
Successfully pushed image-registry.openshift-image-registry.svc:5000/your-project/your-project@sha256:b3c84caa0cadaa4bd2a3a502e7b2fa12b
Push successful

```

Abbildung 1.20: Der Tab „Logs“ auf der Seite „Build Details“

Verwalten von Builds und Bereitstellen mit der Web Console

Die OpenShift Web Console enthält Funktionen zum Verwalten von Builds und Bereitstellungen sowie alle durch die jeweilige Konfiguration ausgelösten einzelnen Builds und Bereitstellungen.

Kapitel 1 | Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster

Ein Großteil dieser Konfiguration erfolgt auf spezifischen Detailseiten für die Builds oder Bereitstellungen, die Sie aktualisieren möchten.

Die Detailseite für eine Build-Konfiguration enthält die Tabs **Details**, **YAML**, **Builds**, **Environment** und **Events** sowie die Schaltfläche **Actions** mit der Aktion **Start Build**. Diese Aktion führt dieselbe Funktion wie der Befehl `oc start-build` aus.

Wenn das Quellcode-Repository der Anwendung nicht für die Verwendung von OpenShift-Webhooks konfiguriert ist, müssen Sie die Web Console oder die CLI verwenden, um neue Builds auszulösen, nachdem Updates an den Quellcode der Anwendung übertragen wurden.

Die Detailseite für eine Bereitstellung bietet wesentlich mehr Funktionalitäten, einschließlich Aktionen zum Anpassen verschiedener Aspekte einer Bereitstellung. Dazu zählt beispielsweise die Anzahl der gewünschten Pods oder der Pod-Storage.

Die Detailseite für eine Bereitstellung enthält auch die Schaltfläche **Action** mit verschiedenen Aktionen, die auf der Seite „Build Config“ ausgeführt werden können. Beispielsweise ist die Aktion **Start Rollout** verfügbar, die dieselbe Funktion wie der Befehl `oc rollout latest` ausführt. Sie müssen die CLI verwenden, um spezifischere `oc rollout`-Vorgänge auszuführen.

Bearbeiten von OpenShift-Ressourcen

Die meisten Ressourcendetailseiten in der OpenShift Web Console enthalten die Schaltfläche **Actions**, über die ein Menü angezeigt wird. Dieses Menü bietet die folgenden Auswahlmöglichkeiten:

- **Edit resource:** Bearbeitet eine bestehende Ressource in der rohen YAML-Syntax in einem Browser-basierten Texteditor mit Syntax-Highlighting. Diese Aktion entspricht der Verwendung des Befehls `oc edit -o yaml`.
- **Delete resource:** Löscht eine vorhandene Ressource. Diese Aktion entspricht der Verwendung des Befehls `oc delete`.
- **Edit Labels:** Öffnet ein modales Dialogfeld zum Bearbeiten von Ressourcenbeschriftungen.
- **Edit Annotations:** öffnet ein modales Dialogfeld zum Bearbeiten der Schlüssel und Werte von Annotationen.

Nicht alle Ressourcenverwaltungsvorgänge können mit der Web Console durchgeführt werden. Vor allem die Aktivitäten des Cluster-Administrators erfordern in der Regel die CLI.



Literaturhinweise

Weitere Informationen über die Organisation, Navigation und Verwendung der Web Console finden Sie im Handbuch *Web Console for Red Hat OpenShift Container Platform 4.6* unter
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/web_console/index

► Angeleitete Übung

Verwalten einer Anwendung mit der Web Console

In dieser Übung verwenden Sie die OpenShift Web Console zum Bereitstellen eines Apache HTTP Server-Container-Images.

Ergebnisse

Sie sollten in der Lage sein, die OpenShift Web Console für Folgendes zu verwenden:

- Erstellen eines neuen Projekts und Hinzufügen einer neuen Anwendung, die ein Container-Image bereitstellt
- Ausführen allgemeiner Fehlerbehebungsaufgaben, wie Anzeigen von Logs, Überprüfen von Ressourcendefinitionen und Löschen von Ressourcen

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen aktiven OpenShift-Cluster
- Auf das Container-Image für die Beispielanwendung (`redhattraining/php-hello-dockerfile`)

Führen Sie den folgenden Befehl auf der `workstation`-VM aus, um die Voraussetzungen zu überprüfen:

```
[student@workstation ~]$ lab deploy-image start
```

Anweisungen

- 1. Öffnen Sie einen Webbrowser, und navigieren Sie zu `https://console-openshift-console.apps.cluster.domain.example.com`, um auf die OpenShift Web Console zuzugreifen. Melden Sie sich an und erstellen Sie ein neues Projekt mit dem Namen `youruser -deploy-image`.
- 1.1. Suchen Sie die Platzhalter-Domain Ihres OpenShift-Clusters. Dies ist die Variable `RHT_OCP4_WILDCARD_DOMAIN` in der Konfigurationsdatei `/usr/local/etc/ocp4.config` des Kursraums.

```
[student@workstation ~]$ grep RHT_OCP4_WILDCARD_DOMAIN /usr/local/etc/ocp4.config  
RHT_OCP4_WILDCARD_DOMAIN=apps.cluster.domain.example.com
```

Eine weitere Möglichkeit, den Hostnamen Ihrer OpenShift Web Console zu ermitteln, besteht in der Überprüfung der Routen im Projekt `openshift-console`. Sie müssen sich zuvor bei OpenShift angemeldet haben.

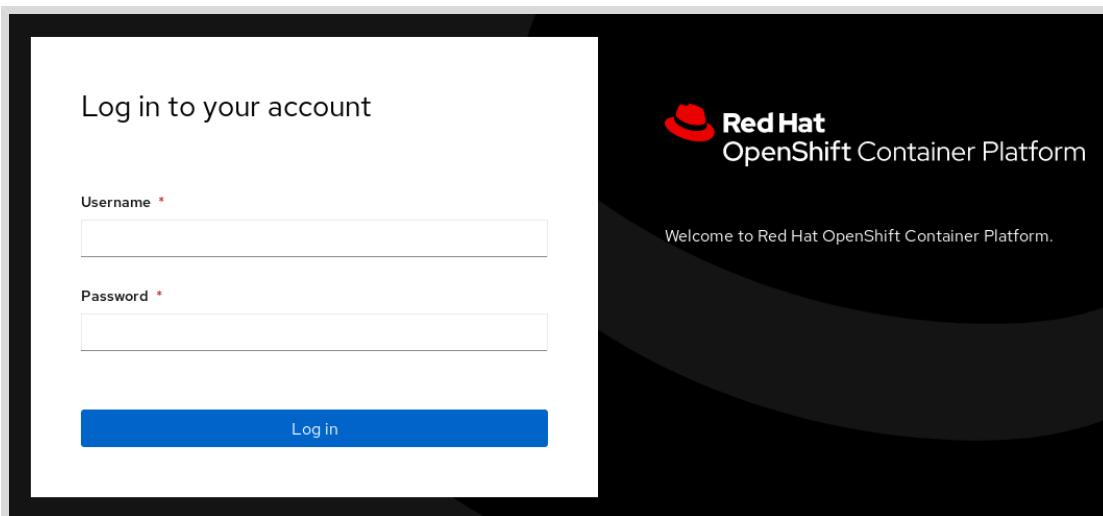
```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
[student@workstation ~]$ oc get route -n openshift-console
NAME      HOST/PORT          PATH ...
console   console-openshift-console.apps.cluster.domain.example.com ...
downloads downloads-openshift-console.apps.cluster.domain.example.com ...
```



Anmerkung

Mit den Standardeinstellungen von Red Hat OpenShift Container Platform können reguläre Benutzer nicht auf das `openshift-console`-Projekt zugreifen, aber die Kursumgebung wurde so konfiguriert, dass diese Berechtigungen erteilt werden.

- 1.2. Öffnen Sie einen Webbrower, und navigieren Sie zu `https://console-openshift-console.apps.cluster.domain.example.com`, um auf die OpenShift Web Console zuzugreifen. Ersetzen Sie `apps.cluster.domain.example.com` durch den Wert, den Sie im vorherigen Schritt erhalten haben. Die Seite zur Anmeldung bei der Web Console sollte angezeigt werden.
- 1.3. Melden Sie sich als Entwicklungsbenutzer an. Ihr Benutzername (`youruser`) ist die Variable `RHT_OCP4_DEV_USER` in der Konfigurationsdatei `/usr/local/etc/ocp4.config` des Kursraums. Ihr Passwort ist der Wert der Variable `RHT_OCP4_DEV_PASSWORD` in derselben Datei.



- 1.4. Navigieren Sie zur Administratoransicht, und wählen Sie dann im Menü auf der linken Seite **Home** → **Projects** aus. Diese Seite wird Benutzern angezeigt, die zum ersten Mal zugreifen. Klicken Sie auf **Create Project**. Geben Sie im Dialogfeld **Create Project** in das Feld **Name** `youruser-deploy-image` ein. Ersetzen Sie `youruser` durch den Wert Ihrer Variable `RHT_OCP4_DEV_USER`. Sie müssen die Felder für den Anzeigenamen und die Beschreibung nicht ausfüllen.

Klicken Sie auf **Create**, um das neue Projekt zu erstellen.

Kapitel 1 | Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster

- 2. Erstellen Sie eine neue Anwendung aus einem vorkonfigurierten Container-Image, das die in PHP geschriebene Anwendung „Hello, World“ enthält, und erstellen Sie eine Route, um die Anwendung öffentlich verfügbar zu machen.

2.1. Klicken Sie auf den Tab **Workloads** auf die Schaltfläche **Container Image**.

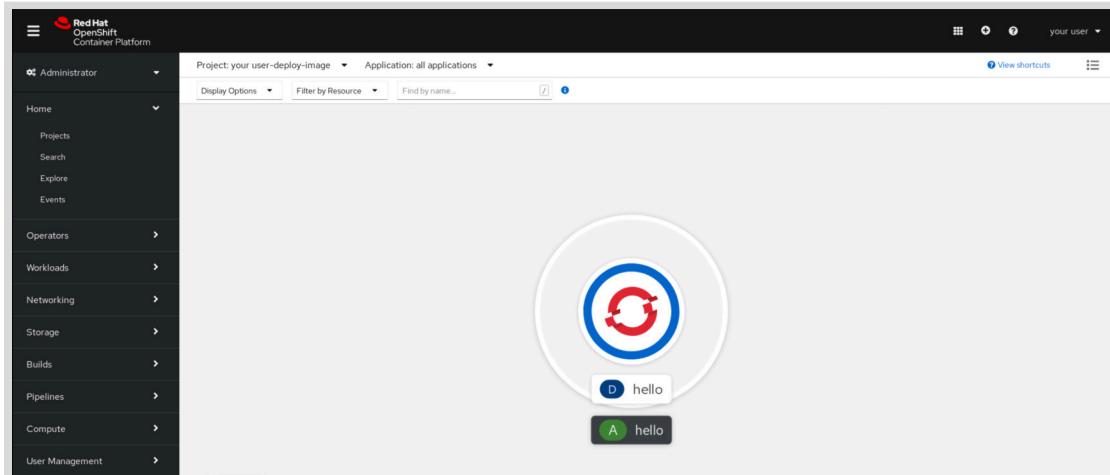
- 2.2. Geben Sie auf der Seite **Deploy Image** im Feld **Image Name** `quay.io/redhattraining/php-hello-dockerfile` ein. Die OpenShift Web Console stellt eine Verbindung zur öffentlichen Registry `quay.io` her und ruft Informationen über das Container-Image ab.

- 2.3. Scrollen Sie nach unten, um Informationen zum Image anzuzeigen, und ersetzen Sie in den Feldern **Application Name** und **Name** `php-hello-dockerfile` durch `hello`.

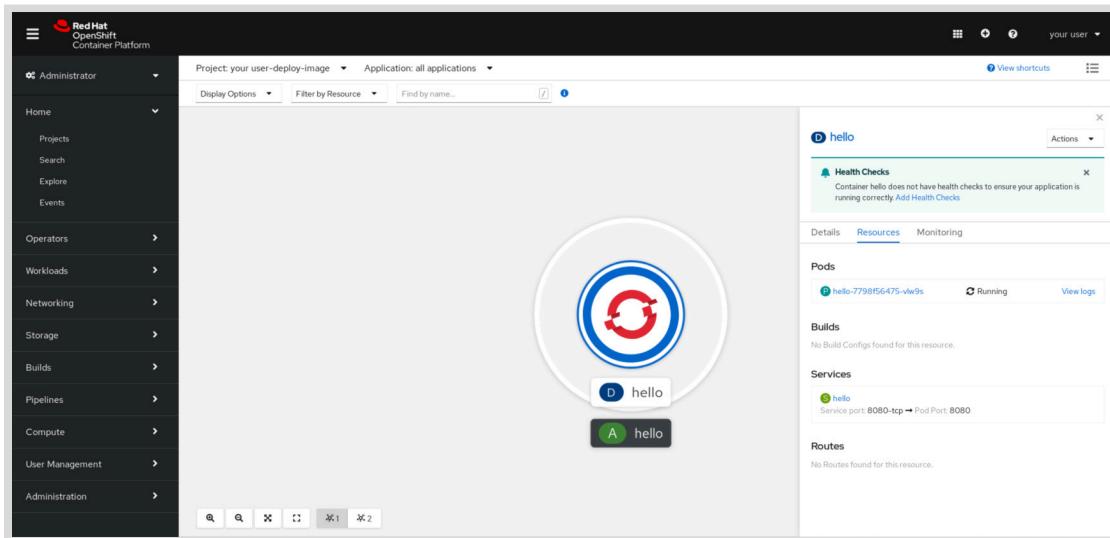
Deaktivieren Sie unten auf der Seite die Option **Create a route to the application** im Abschnitt **Advanced Options**.

Kapitel1 | Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster

- 2.4. Klicken Sie auf **Create**, um die neue Anwendung zu erstellen. Die Web Console erstellt alle OpenShift-Ressourcen, die zum Bereitstellen des Container-Images erforderlich sind, und wechselt zur Seite **Topology** des Projekts.

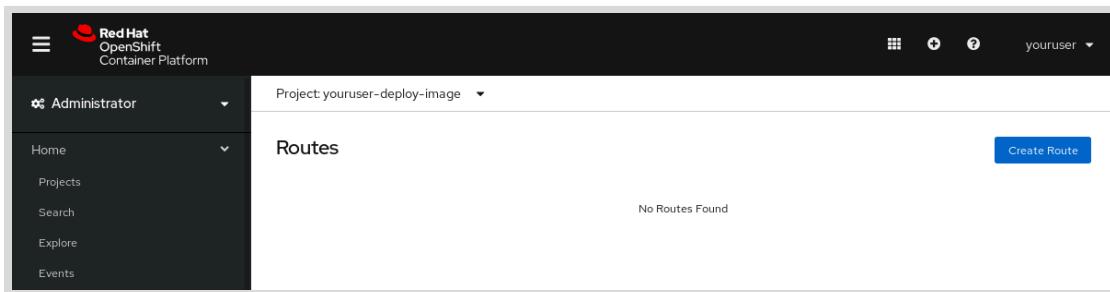


- 2.5. Klicken Sie auf das Symbol über dem Deploymentnamen, um die Deploymentdetails zu erweitern und die Anzahl der ausgeführten Pods anzuzeigen. Warten Sie, bis auf der Übersichtsseite eine erfolgreiche Bereitstellung mit einem Pod angezeigt wird:



- 2.6. Klicken Sie in der Navigationsleiste auf **Networking → Routes**.

Klicken Sie auf der Seite **Routes** auf die Schaltfläche **Create Route**.



Kapitel 1 | Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster

Geben Sie im Feld **Name** `hello-route` ein. Geben Sie `hello-youruser.apps.cluster.domain.example.com` im Feld **Hostname** ein.

Ersetzen Sie `youruser` durch den Wert Ihrer Variable `RHT_OCP4_DEV_USER`.

Das ist der Benutzername, den Sie für die Anmeldung bei OpenShift verwendet haben. Ersetzen Sie `apps.cluster.domain.example.com` durch den Wert der Variable `RHT_OCP4_WILDCARD_DOMAIN`.

The screenshot shows the 'Create Route' page in the OpenShift web interface. The left sidebar has 'Networking' selected under 'Routes'. The main form has 'Name' set to 'hello-route' and 'Hostname' set to 'hello-youruser.apps.cluster.domain.example.com'. There is a link 'Edit YAML' at the top right of the form area.

Scrollen Sie nach unten und wählen Sie den Service **hello** aus der Liste **Service** aus. Wählen Sie in der Liste **Target Port** die Option `8080 → 8080 (TCP)` aus. Ändern Sie die anderen Felder nicht. Scrollen Sie nach unten und klicken Sie auf **Create**.

The screenshot shows the 'Create Route' page with more detailed configuration. The 'Service' dropdown is set to 'hello'. The 'Target Port' dropdown is set to '8080 → 8080 (TCP)'. The 'Path' field contains '/'. The 'Create' and 'Cancel' buttons are at the bottom of the form.

- 2.7. Die Routendetailseite wird geändert, um die URL für den Zugriff auf die Anwendung mit der neuen Route anzuzeigen.

Kapitel 1 | Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster

The screenshot shows the Red Hat OpenShift Container Platform web console. The left sidebar is titled 'Administrator' and has sections for Home, Operators, Workloads, Networking (selected), Services, Routes (selected), Ingresses, Network Policies, Storage, and Builds. The main content area is titled 'Project: youruser-deploy-image' and shows 'Routes > Route Details' for 'hello-route'. The route is listed as 'Accepted' with the URL 'http://hello-youruser.apps.cluster.domain.example.com'. Below this, the 'Route Details' section shows the Name 'hello-route', Namespace 'youruser-deploy-image', Status 'Accepted', Host 'hello-youruser.apps.cluster.domain.example.com', and Path '-'. The 'Details' tab is currently selected.

Klicken Sie auf `http://hello-youruser.apps.cluster.domain.example.com`, um einen neuen Browser-Tab zu öffnen, auf dem die von der PHP-Anwendung zurückgegebene Standardseite angezeigt wird. Das ist eine einfache „Hello, World“-Meldung mit der PHP-Version.

► 3. Untersuchen Sie die Fehlerbehebungsfunktionen der Web Console.

3.1. Zeigen Sie die Logs des Anwendungs-Pods an.

Klicken Sie in der Navigationsleiste auf Workloads → Pods.

The screenshot shows the Red Hat OpenShift Container Platform web console. The left sidebar is titled 'Administrator' and has sections for Home, Operators, Workloads (selected), and Pods. The main content area is titled 'Project: your user-deploy-image' and shows a table of 'Pods'. There is one pod listed: 'hello-7798f56475-vlw9s' with status 'Running'. A 'Create Pod' button is visible at the top right of the table.

Klicken Sie auf den Namen des Anwendungs-Pods, beispielsweise auf `hello-7798f56475-vlw9s`, um die Seite **Pod Details** anzuzeigen. Klicken Sie auf den Tab **Logs**, um die Pod-Logs anzuzeigen. Die Warnung hinsichtlich des vollqualifizierten Domain-Namens des Servers wird erwartet und kann problemlos ignoriert werden.

The screenshot shows the Red Hat OpenShift Container Platform web console. The left sidebar is titled 'Administrator' and has sections for Home, Projects, Search, Explore, Events, Operators, Workloads (selected), and Pods. The main content area is titled 'Project: your user-deploy-image' and shows 'Pods > Pod Details' for 'hello-7798f56475-vlw9s'. The 'Logs' tab is selected. The log output shows several notices about user and group directives being ignored when FPM is not running as root, and a warning about the server's fully qualified domain name. The 'Raw', 'Download', and 'Expand' buttons are visible at the bottom of the log pane.

3.2. Starten Sie eine Shell-Sitzung in einem aktiven Container.

Klicken Sie auf der Seite **Pod Details** auf den Tab **Terminal**, um eine Remote-Shell für den einzigen Container im Anwendungs-Pod zu öffnen. Wenn das Terminalfenster zu klein ist, klicken Sie auf **Expand**, um die Navigationsbereiche der Web Console

Kapitel 1 | Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster

auszublenden. Das Terminal kann nur Befehle ausführen, die im Container-Image der Anwendung vorhanden sind. Der Befehl `ps` ist nicht verfügbar, aber Sie können die ZugriffsLogs von Apache HTTP Server anzeigen.

```
sh-4.4$ ps ax
sh: ps: command not found
sh-4.4$
sh-4.4$ cat /var/log/httpd/access_log
10.131.0.1 - - [04/Aug/2020:08:39:57 +0000] "GET / HTTP/1.1" 200 36 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0"
10.131.0.1 - - [04/Aug/2020:08:39:57 +0000] "GET /favicon.ico HTTP/1.1" 404 209 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0"
sh-4.4$
sh-4.4$ cat /var/log/php-fpm/error.log
[04-Aug-2020 08:10:59] NOTICE: [pool www] 'user' directive is ignored when FPM is not running as root
[04-Aug-2020 08:10:59] NOTICE: [pool www] 'group' directive is ignored when FPM is not running as root
[04-Aug-2020 08:10:59] NOTICE: fpm is running, pid 9
[04-Aug-2020 08:10:59] NOTICE: ready to handle connections
[04-Aug-2020 08:10:59] NOTICE: systemd monitor interval set to 10000ms
sh-4.4$
```

Klicken Sie gegebenenfalls auf **Collapse**, um die Navigationsbereiche der Web Console wieder anzuzeigen.

3.3. Zeigen Sie eine Ressourcendefinition an.

Klicken Sie auf der Seite **Pod Details** auf den Tab **YAML**.

```

1  kind: Pod
2  apiVersion: v1
3  metadata:
4    name: hello-7798f56475
5    annotations:
6      k8s.v1.cni.cncf.io/network-status: |
7        [
8          {
9            "name": "",
10           "interface": "eth0",
11           "ips": [
12             "10.131.0.245"
13           ],
14           "default": true,
15           "dns": {}
16         }
17       k8s.v1.cni.cncf.io/networks-status: |
18        [
19          {
20            "name": "",
21            "interface": "eth0",
22            "ips": [
23              "10.131.0.245"
24            ],
25            "default": true,
26            "dns": {}
27          }
28        ]
29      kubernetes.io/limit-range: |
30        [
31          {
32            "limits": [
33              {
34                "resource": "cpu",
35                "value": "100m"
36              },
37              {
38                "resource": "memory",
39                "value": "100Mi"
40              }
41            ],
42            "hard": [
43              {
44                "resource": "cpu",
45                "value": "100m"
46              },
47              {
48                "resource": "memory",
49                "value": "100Mi"
50              }
51            ]
52          }
53        ]
54      kubernetes.io/taints: |
55        [
56          {
57            "key": "node-role.kubernetes.io/master",
58            "operator": "Exists"
59          }
60        ]
61    ]
62  spec:
63    containers:
64      - name: hello
65        image: hello-world:1.0
66        ports:
67          - containerPort: 80
68        resources:
69          requests:
70            cpu: "100m"
71            memory: "100Mi"
72          limits:
73            cpu: "100m"
74            memory: "100Mi"
75        livenessProbe:
76          httpGet:
77            path: /
78            port: 80
79          initialDelaySeconds: 30
80          periodSeconds: 10
81        readinessProbe:
82          httpGet:
83            path: /
84            port: 80
85          initialDelaySeconds: 30
86          periodSeconds: 10
87        volumeMounts:
88          - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
89            name: default-token-4qj6k
90    volumes:
91      - name: default-token-4qj6k
92        secret:
93          secretName: default-token-4qj6k
94
```

Der Tab enthält einen webbasierten Rich-Texteditor für die YAML-Syntax. Nehmen Sie keine Änderungen vor und klicken Sie auf **Cancel**, um den Editor zu beenden.

► 4. Löschen Sie Ressourcen aus dem Projekt.

- 4.1. Klicken Sie auf der Seite **Pod Details** auf **Actions** → **Delete Pod**. Klicken Sie im Bestätigungsdialogfeld auf **Delete**, um den ausgeführten Pod zu löschen. Die Web Console zeigt die Seite **Pods** an.

Warten Sie bis auf der Seite **Pods** angezeigt wird, dass ein neuer Pod von der Deploymentkonfiguration erstellt wurde, um den gelöschten Pod zu ersetzen.

- 4.2. Klicken Sie in der Navigationsleiste auf **Workloads** → **Deployment**, um die Seite **Deployment** anzuzeigen. Klicken Sie auf **hello**, um die Deployment-Detailseite anzuzeigen.
Klicken Sie in der rechten oberen Ecke auf **Actions** → **Delete Deployment**. Lassen Sie im Bestätigungsdialogfeld das Kontrollkästchen aktiviert, und klicken Sie auf **Delete**, um die Bereitstellung zu löschen.

- 4.3. Klicken Sie in der Navigationsleiste auf **Networking** → **Services**, um zu ermitteln, ob im Projekt weiterhin ein Service vorhanden ist.
Klicken Sie auf **hello**, um die Seite **Service Details** zu öffnen. Klicken Sie in der rechten oberen Ecke auf **Actions** → **Delete Service**. Klicken Sie im Bestätigungsdialogfeld auf **Delete**, um den Service zu löschen.
 - 4.4. Klicken Sie in der Navigationsleiste auf **Networking** → **Routes**, und vergewissern Sie sich, dass weiterhin eine Route im Projekt vorhanden ist.
Klicken Sie auf **hello-route**, um die Seite **Route Details** zu öffnen. Klicken Sie in der rechten oberen Ecke auf **Actions** → **Delete Route**. Klicken Sie im Bestätigungsdialogfeld auf **Delete**, um die Route zu löschen.
- ▶ 5. Löschen Sie das Projekt.
- Klicken Sie in der linken oberen Ecke auf **Home** → **Projects**, um die Seite **Projects** anzuzeigen. Klicken Sie auf das Menüsymbol rechts neben dem Projekt **youruser-deploy-image**, und klicken Sie dann auf **Delete Project**. Geben Sie im Bestätigungsdialogfeld **youruser-deploy-image** ein und klicken Sie dann auf **Delete**, um das Projekt zu löschen.
- Warten Sie, bis das Projekt **youruser-deploy-image** nicht mehr auf der Seite **Projects** angezeigt wird. Beachten Sie, dass es wie im vorherigen Schritt nicht erforderlich ist, Anwendungsressourcen einzeln zu entfernen. Durch das Löschen eines Projekts werden alle Ressourcen im Projekt gelöscht.

Beenden

Führen Sie auf der **workstation** den Befehl **lab deploy-image finish** aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken.

```
[student@workstation ~]$ lab deploy-image finish
```

Hiermit ist die angeleitete Übung beendet.

Verwalten von Anwendungen mit der CLI

Ziele

Nach Abschluss dieses Abschnitts sollten Sie zu Folgendem in der Lage sein:

- Bereitstellen einer Anwendung über den Quellcode und Verwalten der zugehörigen Ressourcen mithilfe der Befehlszeilenschnittstelle
- Beschreiben der zum Verwalten eines Projekts und eines Clusters erforderlichen OpenShift-Rollen
- Beschreiben, wie Source-to-Image das Builder-Image für eine Anwendung bestimmt

OpenShift-Cluster- und Projektverwaltungsberechtigungen

Viele Aufgaben zur Verwaltung eines OpenShift-Clusters erfordern besondere Administratorberechtigungen. Sie haben möglicherweise Zugriff auf einen OpenShift-Cluster, für den Sie als Cluster-Administrator fungieren, aber Sie verfügen normalerweise nicht über diese Zugriffsebene.

OpenShift-Cluster dienen in der Regel vielen verschiedenen Anwendern, möglicherweise aus mehreren Unternehmen. Diese Cluster mit mehreren Knoten bieten Benutzern unterschiedliche Zugriffsebenen: Cluster-Administrator, Projektadministrator und Entwickler.

Mit der Standardkonfiguration für einen OpenShift-Cluster kann ein beliebiger Benutzer neue Projekte erstellen. Ein Benutzer fungiert automatisch als Projektadministrator für die von ihm erstellten Projekte. Ein Cluster-Administrator kann die OpenShift-Cluster-Berechtigungen ändern, damit Benutzer keine Projekte erstellen können.

In diesem Szenario kann nur ein Cluster-Administrator neue Projekte erstellen. Der Cluster-Administrator kann anschließend anderen Benutzern Projektadministrator- und Entwicklerberechtigungen zuweisen.

In der folgenden Liste enthält eine Übersicht der Aufgaben, die Benutzer auf der jeweiligen Zugriffsebene ausführen können:

Cluster-Administrator

Projekte verwalten, Knoten hinzufügen, persistente Volumes erstellen, Projektkontingente zuweisen und andere clusterweite Verwaltungsaufgaben ausführen.

Projektadministrator

Ressourcen in einem Projekt verwalten, Ressourcenbeschränkungen zuweisen und anderen Benutzern die Berechtigung zum Anzeigen und Verwalten von Ressourcen im Projekt gewähren.

Entwickler

Eine Teilmenge der Ressourcen eines Projekts verwalten. Die Teilmenge umfasst die zum Entwickeln und Bereitstellen von Anwendungen erforderlichen Ressourcen. Dazu zählen beispielsweise Build- und Deploymentkonfigurationen, Anforderungen für persistente Volumes, Services, Secrets und Routen. Entwickler können anderen Benutzern keine

Kapitel 1 | Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster

Berechtigung für diese Ressourcen gewähren. Zudem können sie die meisten Ressourcen auf Projektebene wie Ressourcenbeschränkungen nicht verwalten.

Sie führen die meisten praktischen Übungen in diesem Kurs als Benutzer developer durch, der Projektadministratorrechte für die Projekte erhält, die er erstellt. Wenn für eine Übung Cluster-Administratorberechtigungen erforderlich sind, ist in der Übung beschrieben, wie die Anmeldung als Benutzer mit Cluster-Administratorberechtigungen erfolgen soll. Anschließend werden die erforderlichen Verwaltungsaufgaben durchgeführt oder die vorkonfigurierten Administratorressourcen bereitgestellt.



Anmerkung

Der Kurs *OpenShift Enterprise Administration I* (DO280) behandelt, wie Verwaltungsaufgaben durchgeführt werden und wie Benutzern Cluster- und Projektadministratorberechtigungen zugewiesen werden.

Fehlerbehebung bei Builds, Bereitstellungen und Pods mit der CLI

OpenShift bietet drei Hauptmechanismen zum Abrufen von Fehlerbehebungsinformationen zu einem Projekt und dessen Ressourcen:

Statusinformationen

Befehle wie `oc status` und `oc get` liefern zusammenfassende Informationen zu Ressourcen in einem Projekt. Führen Sie diese Befehle aus, um kritische Informationen abzurufen. Dazu zählen beispielsweise, ob ein Build fehlgeschlagen ist oder nicht und ob ein Pod betriebsbereit ist und ausgeführt wird.

Ressourcenbeschreibung

Der Befehl `oc describe` zeigt Details zu einer Ressource, einschließlich ihres aktuellen Zustands, der Konfiguration und aktueller Ereignisse, an. Die Option `-o` mit dem Befehl `oc get` zeigt die vollständige Low-Level-Konfiguration und Statusinformationen zu einer Ressource an. Verwenden Sie diese Befehle zum Überprüfen einer Ressource und zum Ermitteln, ob OpenShift spezifische Fehlerbedingungen erkennen konnte, die mit der Ressource zusammenhängen.

RessourcenLogs

Ausführbare Ressourcen wie Pods und Builds speichern Logs, die mit dem Befehl `oc logs` angezeigt werden können. Diese Logs werden durch die in einem Pod ausgeführte Anwendung oder durch den Build-Prozess generiert. Führen Sie diese Befehle aus, um anwendungsspezifische Fehlermeldungen und Details zu Build-Fehlern abzurufen.

Wenn diese Mechanismen keine ausreichenden Informationen bieten, können Sie die Befehle `oc cp` und `oc rsh` zur direkten Interaktion mit einer containerisierten Anwendung verwenden.

Vergleich zweier Befehle, die Sie zum Beschreiben von OpenShift-Ressourcen verwenden können

Der Befehl `oc describe` kann Beziehungen zwischen Ressourcen nachverfolgen. So werden beispielsweise beim Beschreiben einer Build-Konfiguration Informationen zu den neuesten Builds angezeigt. Der Befehl `oc get -o` zeigt nur Informationen zur angeforderten Ressource an. Wenn der Befehl `oc get -o` beispielsweise für eine Build-Konfiguration ausgeführt wird, werden keine Informationen zu den aktuellen Builds angezeigt.

Führen Sie den Befehl `oc edit` aus, um Änderungen an einer OpenShift-Ressource vorzunehmen. Der Befehl `oc edit` kombiniert das Abrufen der Ressourcenbeschreibung mit dem Befehl `oc get -o` sowie das Öffnen der Ausgabedatei in einem Texteditor und übernimmt dann die Änderungen mit dem Befehl `oc apply`.

Verbessern containerisierter AnwendungsLogs

Die Benutzerfreundlichkeit der in OpenShift gespeicherten AnwendungsLogs hängt vom Design des Anwendungscontainer-Images ab. Eine containerisierte Anwendung soll die gesamte Logausgabe an die Standardausgabe senden. Wenn die Anwendung ihre Logausgabe an eine Logdateien sendet, was bei nicht containerisierten Anwendungen üblich ist, werden diese Logs im temporären Storage des Containers gespeichert und gehen verloren, wenn der Anwendungs-Pod beendet wird.

OpenShift stellt darüber hinaus ein optionales Logierungs-Subsystem bereit, das auf dem EFK-Stack (Elasticsearch, Fluentd und Kibana) basiert. Das Logierungs-Subsystem bietet einen langfristigen Storage und Suchfunktionen für OpenShift-Cluster-Knoten und -AnwendungsLogs. Eine Anwendung kann so konzipiert werden, dass sie das OpenShift-Logierungs-Subsystem vollständig nutzt oder dass sie ihre Logausgabe an die Standardausgabe sendet und dem EFK-Stack ermöglicht, die zugehörigen Logs zu erfassen und zu verarbeiten.

Die Installation und Konfiguration des OpenShift-Logierungs-Subsystems wird in diesem Kurs nicht behandelt.

Lesen von Build-Logs

Build-Logs für einen spezifischen Build können auf zwei Arten abgerufen werden: Sie können auf die Build-Konfiguration oder die Build-Ressource verweisen oder Sie können auf den Build-Pod verweisen.

Im folgenden Beispiel wird eine Build-Konfiguration mit dem Namen `myapp` verwendet:

```
[user@host ~]$ oc logs bc/myapp
```

Die Logs aus einer Build-Konfiguration sind die Logs aus dem neuesten Build, unabhängig davon, ob er erfolgreich war oder nicht.

In diesem Beispiel wird der zweite Build aus derselben Build-Konfiguration verwendet:

```
[user@host ~]$ oc logs build/myapp-2
```

In diesem Beispiel wird der zum Ausführen desselben Builds erstellte Build-Pod verwendet:

```
[user@host ~]$ oc logs myapp-build-2
```

Erlangen des direkter Zugriffs auf eine containerisierte Anwendung

Wenn eine Anwendung ihre Logs im temporären Container-Storage speichert, führen Sie die Befehle `oc cp` und `oc rsync` aus, um die Logdateien abzurufen. Diese Befehle können verwendet werden, um eine beliebige Datei aus einem aktiven Containerdateisystem abzurufen. Dazu zählen beispielsweise Konfigurationsdateien für die containerisierte Anwendung.

Kapitel 1 | Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster

Sie müssen den Remote-Dateipfad im Container-Dateisystem mit den Befehlen `oc cp` und `oc rsync` verwenden. Sie können die Dateien im temporären Container-Storage oder auf einem persistenten Volume, das durch den Container gemountet wird, speichern.

Führen Sie den folgenden Befehl aus, um beispielsweise die in einem Anwendungs-Pod mit dem Namen `frontend` gespeicherten Apache HTTP Server-FehlerLogs abzurufen.

```
[user@host ~]$ oc cp frontend-1-zvjhbt:/var/log/httpd/error_log \
/tmp/frontend-server.log
```

Der Befehl `oc cp` kopiert standardmäßig die gesamten Ordner. Wenn das Quellargument eine einzelne Datei ist, muss das Zielargument ebenfalls eine einzelne Datei sein. Im Gegensatz zum UNIX-Befehl `cp` kann der Befehl `oc cp` keine Quelldatei in einen Zielordner kopieren.

Damit der Befehl `oc cp` funktioniert, muss das zugrunde liegende Anwendungscontainer-Image den Befehl `tar` bereitstellen. Wenn `tar` nicht im Anwendungscontainer installiert ist, schlägt `oc cp` fehl.

Derselbe Befehl wird verwendet, um Dateien in ein Containerdateisystem zu kopieren. Verwenden Sie diese Funktion, um Schnelltests in einem aktiven Container durchzuführen. Verwenden Sie diese Funktion nicht, um ein Problem dauerhaft zu beheben. Zum Lösen eines Problems in einem Container wird empfohlen, die Korrektur auf das Container-Image und auf die Anwendungsressourcen anzuwenden und anschließend einen neuen Anwendungs-Pod bereitzustellen.

Der Befehl `oc rsync` synchronisiert lokale Ordner mit Remote-Ordnern aus einem aktiven Container. Er verwendet den lokalen Befehl `rsync` zum Reduzieren der Bandbreitennutzung, erfordert jedoch nicht die Verwendung des Befehls `rsync` oder `ssh`, um im Container-Image verfügbar zu sein.

Wenn das Abrufen von Dateien nicht ausreichend ist, um die Fehlerbehebung für einen aktiven Container vorzunehmen, wird mit dem Befehl `oc rsh` eine Remote-Shell erstellt, um Befehle im Container auszuführen. Er verwendet die OpenShift-Master-API, um einen sicheren Tunnel zum Remote-Pod zu erstellen. Er verwendet jedoch weder den `ssh`- noch den `rsh`-UNIX-Befehl.

Im folgenden Beispiel wird das Ausführen des Befehls `ps ax` in einem Pod mit dem Namen `frontend` veranschaulicht:

```
[user@host ~]$ oc rsh frontend-1-zvjhbt ps ax
```



Anmerkung

Viele Container-Images enthalten keine allgemeinen UNIX-Fehlerbehebungsbefehle wie `ps` und `ping`. Der Befehl `oc rsh` kann nur Befehle ausführen, die vom Remote-Container bereitgestellt werden.

Fügen Sie dem Befehl `-t` die Option `oc rsh` hinzu, um eine interaktive Shell-Sitzung im Container zu starten:

```
[user@host ~]$ oc rsh -t frontend-1-zvjhbt
```



Anmerkung

Die vom Befehl `oc rsh` angezeigte Shell-Eingabeaufforderung hängt von der Shell ab, die vom Container-Image bereitgestellt wurde.

Umgebungsvariablen für Build und Bereitstellung

Viele Container-Images erwarten, dass Benutzer Umgebungsvariablen zum Bereitstellen von Konfigurationsinformationen definieren. Beispielsweise ist für das MySQL-Datenbank-Image aus dem Red Hat Container Catalog die Variable `MYSQL_DATABASE` erforderlich, um den Datenbanknamen anzugeben.

Fügen Sie dem Befehl `oc new-app` die Option `-e` hinzu, um Werte für Umgebungsvariablen bereitzustellen. Diese Werte werden in der Deploymentkonfiguration gespeichert und allen Pods hinzugefügt, die durch eine Bereitstellung erstellt wurden.

Source-to-Image (S2I)-Builder-Images können auch Konfigurationsparameter aus Umgebungsvariablen akzeptieren. Beispielsweise benötigen Node.js-Anwendungen häufig ein NPM-Repository, den primären Paket-Manager für Node.js, um die von der Anwendung benötigten Node.js-Abhängigkeiten herunterzuladen. Aus diesem Grund akzeptiert der Node.js-S2I-Builder aus dem Container Catalog entweder die Variable `npm_config_registry` oder `NPM_MIRROR`, um eine URL bereitzustellen, über die der S2I-Builder das NPM-Modul-Repository finden kann, das zum Abrufen der benötigten Node.js-Abhängigkeiten erforderlich ist.



Anmerkung

Für den Befehl `npm`, der vom Node.js-S2I-Builder-Image ausgeführt wird, müssen Sie einen Wert für die Umgebungsvariable `npm_config_registry` angeben.

Für das Skript `assemble` aus dem Node.js-S2I-Builder-Image, das den Befehl `npm` aufruft, müssen Sie einen Wert für die Umgebungsvariable `NPM_MIRROR` angeben.

Mit S2I-Builder-Image-Variablen kann verhindert werden, dass Konfigurationsinformationen mit Anwendungsquellen im Git-Repository gespeichert werden. Für verschiedene Umgebungen könnten unterschiedliche Konfigurationen erforderlich sein, beispielsweise:

- Eine Entwicklungsumgebung würde einen NPM-Repository-Server verwenden, auf dem Entwickler neue Module installieren können.
- Eine QS-Umgebung würde einen anderen NPM-Repository-Server verwenden, auf dem ein Sicherheitsteam Module überprüfen könnte, bevor sie in höhere Umgebungen befördert werden.

Sie definieren Umgebungsvariablen für einen Anwendungs-Pod mit der Option `-e` des Befehls `oc new-app`. Für einen Builder-Pod definieren Sie die Umgebungsvariablen mit der Option `--build-env` des Befehls `oc new-app`.

Beachten Sie, dass eine Deploymentkonfiguration die Umgebungsvariablen für Anwendungs-Pods speichert. Demgegenüber speichert eine Build-Konfiguration die Umgebungsvariablen für Builder-Pods. Sehen Sie in der Dokumentation für das jeweilige Builder-Image nach, um Informationen zu dessen Variablen und Standardwerten zu erhalten.



Literaturhinweise

Weitere Informationen finden Sie im Kapitel *Understanding Containers, Images, and Imagestreams* des Handbuchs *Images* für Red Hat OpenShift Container Platform 4.6 unter

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/images/index#understanding-images

► Angeleitete Übung

Verwalten einer Anwendung mit der CLI

In dieser Übung stellen Sie eine containerisierte Anwendung bereit, die aus mehreren Pods aus einer Vorlage besteht. Zudem nehmen Sie eine Fehlerbehebung vor und korrigieren einen Deploymentfehler.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Erstellen einer Anwendung mit einer benutzerdefinierten Vorlage. Die Vorlage stellt einen Anwendungs-Pod anhand von PHP-Quellcode und einen Datenbank-Pod anhand eines MySQL-Servercontainer-Images bereit.
- Ermitteln der Ursache eines Deploymentfehlers mit der OpenShift-CLI
- Beheben des Fehlers mit der OpenShift-CLI

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen aktiven OpenShift-Cluster
- Auf das S2I-Builder-Image und das Datenbank-Image, die von der benutzerdefinierten Vorlage (PHP 7.2 und MySQL 5.7) benötigt werden
- Auf die Beispielanwendung im Git-Repository (quotes)

Führen Sie den folgenden Befehl auf der workstation-VM aus, um die Voraussetzungen zu überprüfen und die Dateien herunterzuladen, die zum Durchführen dieser Übung erforderlich sind:

```
[student@workstation ~]$ lab build-template start
```

Anweisungen

- 1. Überprüfen Sie den Quellcode der Quotes-Anwendung.
- 1.1. Wechseln Sie zu Ihrem lokalen Klon des D0288-apps-Git-Repository und checken Sie den main-Branch des Kurs-Repository aus, um sicherzustellen, dass Sie diese Übung von einem als fehlerfrei bekannten Zustand aus starten:

```
[student@workstation ~]$ cd ~/D0288-apps
[student@workstation D0288-apps]$ git checkout main
...output omitted...
```

- 1.2. Die Anwendung besteht aus zwei PHP-Seiten:

```
[student@workstation D0288-apps]$ ls ~/D0288-apps/quotes
get.php index.php
```

Auf der Willkommensseite (`index.php`) wird eine kurze Beschreibung der Anwendung angezeigt. Sie verknüpft zu einer weiteren Seite (`get.php`), die ein zufälliges Kontingent aus einer MySQL-Datenbank erhält.

- 1.3. Überprüfen Sie den PHP-Code, der auf die Datenbank zugreift:

```
[student@workstation D0288-apps]$ less ~/D0288-apps/quotes/get.php
<?php
$link = mysqli_connect($_ENV["DATABASE_SERVICE_NAME"], $_ENV["DATABASE_USER"],
$_ENV["DATABASE_PASSWORD"], $_ENV["DATABASE_NAME"]);
if (!$link) {
    http_response_code(500);
    error_log("Error: unable to connect to database\n");
    die();
}
...output omitted...
```

Drücken Sie zum Beenden die Taste q.

Die Beispielanwendung verwendet nur Standard-PHP-Funktionen, aber kein Framework. Die Anwendung verwendet Umgebungsvariablen zum Abrufen von Datenbankverbindungsparametern und gibt bei Fehlern HTTP-Standardstatuscodes zurück.

- 2. Überprüfen Sie die benutzerdefinierte Vorlage unter `~/D0288/labs/build-template/php-mysql-ephemeral.json`. Der vollständige Inhalt der benutzerdefinierten Vorlage ist aus Platzgründen nicht aufgeführt. Sie müssen keine Änderungen an der Vorlage vornehmen. Sie ist einsatzbereit. Die folgenden Auflistungen zeigen die wichtigsten Teile der Vorlage:

- 2.1. Die Vorlage beginnt mit der Definition eines Secrets und einer Route:

```
{
  "kind": "Template",
  "apiVersion": "'template.openshift.io/v1",
  "metadata": {
    "name": "php-mysql-ephemeral", ①
  ...output omitted...
  "objects": [
    {
      "apiVersion": "v1",
      "kind": "Secret", ②
  ...output omitted...
    },
    "stringData": {
      "database-password": "${DATABASE_PASSWORD}",
      "database-user": "${DATABASE_USER}"
  ...output omitted...
    {
      "apiVersion": "route.openshift.io/v1",
      "kind": "Route", ③
    }
  ]
```

```
...output omitted...
  "spec": {
    "host": "${APPLICATION_DOMAIN}",
    "to": {
      "kind": "Service",
      "name": "${NAME}"
    }
  }
...output omitted...
```

- ❶ Die Vorlage ist eine Kopie der Standardvorlage `cakephp-mysql-example` mit Ressourcen und Parametern, die für das gelöschte Framework spezifisch sind. Die benutzerdefinierte Vorlage eignet sich für eine einfache PHP-Anwendung, die eine MySQL-Datenbank verwendet.
- ❷ Ein Secret speichert Datenbankanmeldedaten und füllt Umgebungsvariablen in den Anwendungs- und Datenbank-Pods. OpenShift-Secrets werden später in diesem Buch erläutert.
- ❸ Eine Route bietet externen Zugriff auf die Anwendung.

- 2.2. Die Vorlage definiert Ressourcen für eine PHP-Anwendung. In der folgenden Auflistung ist die Build-Konfiguration dargestellt. Die Service- und Image-Stream-Ressourcen wurden weggelassen:

```
...output omitted...
{
  "apiVersion": "build.openshift.io/v1",
  "kind": "BuildConfig", ❶
...output omitted...
  "source": {
    "contextDir": "${CONTEXT_DIR}",
    "git": {
      "ref": "${SOURCE_REPOSITORY_REF}",
      "uri": "${SOURCE_REPOSITORY_URL}"
    },
    "type": "Git"
  },
  "strategy": {
    "sourceStrategy": {
      "from": {
        "kind": "ImageStreamTag", ❷
        "name": "php:7.3-ubi8",
      }
    }
  }
...output omitted...
```

- ❶ Eine Build-Konfiguration verwendet den S2I-Prozess zum Erstellen und Bereitstellen einer PHP-Anwendung anhand des Quellcodes. Die Build-Konfiguration und die zugehörigen Ressourcen entsprechen denen, die über den Befehl `oc new-app` für das Git-Repository erstellt würden.
- ❷ Ein standardmäßiger OpenShift-Image-Stream stellt das PHP-Laufzeit-Build-Image bereit.

- 2.3. Die Vorlage definiert Ressourcen für eine MySQL-Datenbank. In der folgenden Auflistung ist die Bereitstellung dargestellt. Die Service- und Image-Stream-Ressourcen wurden weggelassen:

```

...output omitted...
{
    "apiVersion": "apps/v1",
    "kind": "Deployment", ①
...output omitted...
    "containers": [
...output omitted...
        "name": "mysql",
        "ports": [
            {
                "containerPort": 3306
...output omitted...
        "volumes": [
            {
                "emptyDir": {}, ②
                "name": "data"
...output omitted...
        "triggers": [
...output omitted...
        "env": {
...output omitted...
            "image": "mysql:5.7", ③
...output omitted...

```

- ① Eine Bereitstellung stellt einen MySQL-Datenbankcontainer bereit. Die Bereitstellung und die zugehörigen Ressourcen entsprechen denen, die über den Befehl `oc new-app` für das Datenbank-Image erstellt würden.
- ② Die Datenbank ist nicht durch persistenten Storage gesichert. Beim Neustart des Datenbank-Pods würden alle Daten verloren gehen.
- ③ Ein standardmäßiger OpenShift-Image-Stream stellt das MySQL-Datenbank-Image bereit.

2.4. Abschließend werden in der Vorlage einige Parameter definiert. Einige dieser Parameter sind unten aufgeführt. Sie verwenden später mehrere davon.

```

...output omitted...
"parameters": [
{
    "name": "NAME",
    "displayName": "Name",
    "description": "The name assigned to all of the app objects defined in
this template.",
...output omitted...
{
    "name": "SOURCE_REPOSITORY_URL", ①
    "displayName": "Git Repository URL",
    "description": "The URL of the repository with your application source
code.",
...output omitted...
{

```

```
"name": "DATABASE_USER", ②  
"displayName": "Database User",  
...output omitted...
```

- ① Parameter stellen anwendungsspezifische Konfigurationen wie die Git-Repository-URL bereit.
- ② Die Parameter der Vorlage umfassen auch die Datenbankkonfiguration und die Anmeldedaten für die Verbindung.

► 3. Installieren Sie die benutzerdefinierte Vorlage.

3.1. Laden Sie die Konfiguration Ihrer Kursumgebung.

Führen Sie den folgenden Befehl aus, um die in der ersten angeleiteten Übung erstellten Umgebungsvariablen zu laden:

```
[student@workstation D0288-apps]$ source /usr/local/etc/ocp4.config
```

3.2. Melden Sie sich bei OpenShift mit Ihrem Entwicklerbenutzernamen an:

```
[student@workstation D0288-apps]$ oc login -u ${RHT_OCP4_DEV_USER} \  
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}  
Login successful.  
...output omitted...
```

3.3. Suchen Sie nach einer Vorlage, die PHP und MySQL verwendet. OpenShift stellt eine Vorlage auf Grundlage des CakePHP-Frameworks bereit.

Diese Vorlage eignet sich nicht für die Anwendung „Quotes“, da sie Ressourcen und Abhängigkeiten hinzufügt, die das Framework benötigt. Für diese Übung ist eine einfachere Vorlage bereitgestellt und zwar die Vorlage, die Sie im vorherigen Schritt überprüft haben.

```
[student@workstation D0288-apps]$ oc get templates -n openshift | grep php \  
| grep mysql  
cakephp-mysql-example      An example CakePHP application ...  
cakephp-mysql-persistent    An example CakePHP application ...
```

3.4. Erstellen Sie ein neues Projekt zum Hosten der benutzerdefinierten Vorlage:

```
[student@workstation D0288-apps]$ oc new-project ${RHT_OCP4_DEV_USER}-common  
Now using project "youruser-common" on server  
"https://api.cluster.domain.example.com:6443".  
...output omitted...  
[student@workstation D0288-apps]$ oc create -f \  
~/D0288/labs/build-template/php-mysql-ephemeral.json  
template.template.openshift.io/php-mysql-ephemeral created
```

Red Hat empfiehlt, dass Sie wiederverwendbare OpenShift-Ressourcen, beispielsweise Image-Streams und Vorlagen, in einem gemeinsamen Projekt erstellen.

► 4. Stellen Sie die Anwendung mit der benutzerdefinierten Vorlage bereit.

4.1. Erstellen Sie ein neues Projekt zum Hosten der Anwendung:

```
[student@workstation D0288-apps]$ oc new-project \
${RHT_OCP4_DEV_USER}-build-template
Now using project "youruser-build-template" on server
"https://api.cluster.domain.example.com:6443".
...output omitted...
```

4.2. Überprüfen Sie die Vorlagenparameter, um zu entscheiden, welche zum Bereitstellen der Anwendung „Quotes“ erforderlich sind. Lesen Sie die Beschreibung der Parameter der Vorlage:

```
[student@workstation D0288-apps]$ oc describe template php-mysql-ephemeral \
-n ${RHT_OCP4_DEV_USER}-common
Name: php-mysql-ephemeral
...output omitted...
Parameters:
  Name:          NAME
  Display Name: Name
  Description:   The name assigned to all of the app objects defined in this
template.
  Required:      true
  Value:         php-app
...output omitted...
```

4.3. Überprüfen Sie das Skript `create-app.sh`. Es enthält den Befehl `oc new-app`, der die benutzerdefinierte Vorlage verwendet und alle zum Bereitstellen der Anwendung „Quotes“ erforderlichen Parameter bereitstellt. Auf diese Weise müssen Sie keinen langen Befehl eingeben:

```
[student@workstation D0288-apps]$ cat ~/D0288/labs/build-template/create-app.sh
...output omitted...
oc new-app --template ${RHT_OCP4_DEV_USER}-common/php-mysql-ephemeral \
-p NAME=quotesapi \
-p APPLICATION_DOMAIN=quote-${RHT_OCP4_DEV_USER}.${RHT_OCP4_WILDCARD_DOMAIN} \
-p SOURCE_REPOSITORY_URL=https://github.com/${RHT_OCP4_GITHUB_USER}/D0288-apps \
-p CONTEXT_DIR=quotes \
-p DATABASE_SERVICE_NAME=quotesdb \
-p DATABASE_USER=user1 \
-p DATABASE_PASSWORD=mypa55 \
--name quotes
```

4.4. Führen Sie das Skript `create-app.sh` aus:

```
[student@workstation D0288-apps]$ ~/D0288/labs/build-template/create-app.sh
--> Deploying template "youruser-common/php-mysql-ephemeral" for "youruser-common/
php-mysql-ephemeral" to project youruser-build-template
...output omitted...
--> Creating resources ...
  secret "quotesapi" created
  service "quotesapi" created
  route.route.openshift.io "quotesapi" created
  imagestream.image.openshift.io "quotesapi" created
```

```
buildconfig.build.openshift.io "quotesapi" created
deploymentconfig.apps.openshift.io "quotesapi" created
service "quotesdb" created
deploymentconfig.apps.openshift.io "quotesdb" created
--> Success
...output omitted...
```

- 4.5. Verfolgen Sie die Build-Logs der Anwendung:

```
[student@workstation D0288-apps]$ oc logs -f bc/quotesapi
Cloning "https://github.com/youruser/D0288-apps" ...
...output omitted...
Push successful
```

- 4.6. Warten Sie, bis die Anwendungs- und Datenbank-Pods bereit sind und ausgeführt werden. Beachten Sie, dass die genauen Namen Ihrer Pods von denen in der folgenden Ausgabe abweichen können:

```
[student@workstation D0288-apps]$ oc get pod
NAME           READY   STATUS    RESTARTS   AGE
quotesapi-1-build   0/1     Completed   0          89s
quotesapi-7d76ff58f8-6j2gx   1/1     Running    0          28s
quotesdb-6b7ffcc649-ds1pq   1/1     Running    0          80s
```

Notieren Sie sich die Namen der Anwendungs- und Datenbank-Pods (quotesapi-7d76ff58f8-6j2gx und quotesdb-6b7ffcc649-ds1pq in der Beispieldaten). Sie benötigen diese für die nächsten Schritte.

- 4.7. Verwenden Sie die Route, die von der Vorlage erstellt wurde, um den Endpunkt der Anwendung /get.php zu testen. Es wird ein HTTP-Fehlercode zurückgegeben:

```
[student@workstation D0288-apps]$ oc get route
NAME      HOST/PORT
quotesapi  quote-youruser.apps.cluster.domain.example.com  ...
[student@workstation D0288-apps]$ curl -si \
  http://quote-$RHT_OCP4_DEV_USER.$RHT_OCP4_WILDCARD_DOMAIN/get.php
HTTP/1.1 500 Internal Server Error
...output omitted...
```

- 5. Führen Sie eine Fehlerbehebung hinsichtlich der Konnektivität zwischen dem Anwendungs- und Datenbank-Pod durch.

Die Anwendung wird nicht erwartungsgemäß ausgeführt. Die Build- und Deploymentprozesse waren jedoch erfolgreich. Der Anwendungsfehler liegt möglicherweise an einem Anwendungs-Bug, an einer nicht erfüllten Voraussetzung oder an einer falschen Konfiguration.

Überprüfen Sie als ersten Fehlerbehebungsschritt, ob der Anwendungs-Pod eine Verbindung zum richtigen Datenbank-Pod herstellt.

- 5.1. Überprüfen Sie, ob der Datenbankservice den richtigen Datenbank-Pod gefunden hat. Verwenden Sie den Namen des Datenbank-Pod, den Sie in Schritt 4.6 erhalten haben:

```
[student@workstation D0288-apps]$ oc describe svc quotesdb | grep Endpoints  
Endpoints: 10.129.0.142:3306  
[student@workstation ~]$ oc describe pod quotesdb-6b7ffcc649-ds1pq | grep IP  
IP: 10.129.0.142
```

- 5.2. Überprüfen Sie die Anmeldedaten des Datenbank-Pod:

```
[student@workstation D0288-apps]$ oc describe pod quotesapi-7d76ff58f8-6j2gx \  
| grep -A 4 Environment  
Environment:  
  MYSQL_USER: <set to the key 'database-user' in secret 'quotesapi'>  
  MYSQL_ROOT_PASSWORD: <set to the key 'database-password' in secret  
'quotesapi'>  
  MYSQL_PASSWORD: <set to the key 'database-password' in secret 'quotesapi'>  
  MYSQL_DATABASE: phpapp  
Mounts:
```

- 5.3. Überprüfen Sie die Datenbankverbindungsparameter im Anwendungs-Pod.

Verwenden Sie den Namen des Anwendungs-Pod, den Sie von Schritt 4.6 erhalten haben:

```
[student@workstation D0288-apps]$ oc describe pod quotesapi-7d76ff58f8-6j2gx \  
| grep -A 5 Environment  
Environment:  
  DATABASE_SERVICE_NAME: quotesdb  
  DATABASE_NAME: phpapp  
  DATABASE_USER: <set to the key 'database-user' in secret  
'quotesapi'>  
  DATABASE_PASSWORD: <set to the key 'database-password' in secret  
'quotesapi'>  
Mounts:
```

Beachten Sie, dass die Umgebungsvariablen, die die Datenbankanmeldedaten bereitstellen, für beide Pods identisch sind:

- DATABASE_NAME entspricht dem Wert von MYSQL_DATABASE.
 - DATABASE_USER ist auf den Wert des Schlüssels database-user im Secret quotesapi festgelegt.
 - DATABASE_PASSWORD ist auf den Wert des Schlüssels database-user im Secret quotesapi festgelegt.
 - DATABASE_SERVICE_NAME entspricht dem Namen des Datenbankservice, also quotesdb.
- 5.4. Überprüfen Sie, ob der Anwendungs-Pod auf den Datenbank-Pod zugreifen kann. Das PHP-S2I-Builder-Image stellt keine allgemeinen Netzwerk-Dienstprogramme wie den Befehl ping bereit. In diesem Fall liefert der Befehl echo jedoch eine nützliche Ausgabe:

```
[student@workstation D0288-apps]$ oc rsh quotesapi-7d76ff58f8-6j2gx bash -c \  
'echo > /dev/tcp/$DATABASE_SERVICE_NAME/3306 && echo OK || echo FAIL'  
OK
```

Die Ausgabe des vorherigen Befehls weist darauf hin, dass die Netzwerkverbindung nicht das Problem ist.

- 6. Überprüfen Sie die AnwendungsLogs, um die Ursache des Fehlers herauszufinden und ihn zu beheben.

6.1. Überprüfen Sie die AnwendungsLogs.

```
[student@workstation D0288-apps]$ oc logs quotesapi-7d76ff58f8-6j2gx  
AH00558: httpd: Could not reliably determine the server's fully qualified domain  
name, using 10.129.0.143. Set the 'ServerName' directive globally to suppress  
this message  
...output omitted...  
[Mon May 27 14:54:56.187516 2019] [php7:notice] [pid 52] [client  
10.128.2.3:43952] SQL error: Table 'phpapp.quote' doesn't exist\n  
10.128.2.3 - - [27/May/2019:14:54:51 +0000] "GET /get.php HTTP/1.1" 500 - "-"  
"curl/7.29.0"  
...output omitted...
```

Die Meldung zu den Servernamen kann problemlos ignoriert werden. Der Logeintrag vor dem HTTP-Fehlercode zeigt einen SQL-Fehler an. Der SQL-Fehler gibt an, dass die Anwendung die Tabelle `quote` in der Datenbank `phpapp` nicht abfragen kann.

Im vorherigen Schritt wurde angegeben, dass der Datenbankname richtig ist. Die logische Schlussfolgerung lautet, dass die Tabelle nicht erstellt wurde. Ein SQL-Skript zum Füllen der Datenbank befindet sich als Teil der Dateien dieser Übung im Ordner `~/D0288/labs/build-template`.

6.2. Kopieren Sie das SQL-Skript in den Datenbank-Pod:

```
[student@workstation D0288-apps]$ oc cp ~/D0288/labs/build-template/quote.sql \  
quotesdb-6b7ffcc649-ds1pq:/tmp/quote.sql
```

6.3. Führen Sie das SQL-Skript im Datenbank-Pod aus:

```
[student@workstation D0288-apps]$ oc rsh -t quotesdb-6b7ffcc649-ds1pq  
sh-4.2$ mysql -u$MYSQL_USER -p$MYSQL_PASSWORD $MYSQL_DATABASE < /tmp/quote.sql  
...output omitted...  
sh-4.2$ exit  
[student@workstation D0288-apps]$
```

6.4. Führen Sie die Anwendung aus, um zu überprüfen, ob sie jetzt funktioniert: Sie erhalten ein zufälliges Zitat. Denken Sie daran, den Namen des Route-Hosts aus Schritt 4.6 zu verwenden:

```
[student@workstation DO288-apps]$ curl -si \
http://quote-$RHT_OCP4_DEV_USER.$RHT_OCP4_WILDCARD_DOMAIN/get.php
HTTP/1.1 200 OK
...output omitted...
Always remember that you are absolutely unique. Just like everyone else.
```

Es wird wahrscheinlich eine andere Zufallsmeldung angezeigt. Der Erhalt eines Zitats belegt jedoch, dass die Anwendung jetzt funktioniert.

- 7. Führen Sie eine Bereinigung durch. Wechseln Sie in Ihren Benutzerordner und löschen Sie die in dieser Übung erstellten Projekte.

```
[student@workstation DO288-apps]$ cd ~
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-build-template
project.project.openshift.io "youruser-build-template" deleted
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-common
project.project.openshift.io "youruser-common" deleted
```

Beenden

Führen Sie auf der workstation den Befehl `lab build-template finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken.

```
[student@workstation ~]$ lab build-template finish
```

Hiermit ist die angeleitete Übung beendet.

► Praktische Übung

Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster

In dieser praktischen Übung stellen Sie eine Anwendung für ein OpenShift-Cluster anhand des Quellcodes bereit. Die Build-Konfigurationsdatei der Anwendung weist einen Fehler auf, den Sie diagnostizieren und beheben sollen.



Anmerkung

Für den Befehl `grade` am Ende jeder praktischen Übung eines Kapitels ist es erforderlich, dass Sie die exakten Projektnamen und anderen Identifikatoren, wie in der Spezifikation der praktischen Übung angegeben, verwenden.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Erstellen einer Anwendung mit der Quellcode-Build-Strategie
- Überprüfen der Build-Logs zum Suchen von Informationen zu einem Build-Fehler
- Aktualisieren der Build-Tool-Konfiguration der Anwendung zum Beheben des Build-Fehlers
- Erneutes Erstellen der Anwendung und Verifizieren, ob sie erfolgreich bereitgestellt wurde

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen aktiven OpenShift-Cluster
- Auf das S2I-Builder-Image für Node.js 12-Anwendungen
- Auf die Anwendung im Git-Repository (`nodejs-helloworld`)

Führen Sie den folgenden Befehl auf `workstation` aus, um die Voraussetzungen zu überprüfen. Der Befehl lädt auch die Hilfs- und Lösungsdateien für die Wiederholungsübung herunter:

```
[student@workstation ~]$ lab source-build start
```

Anforderungen

Die bereitgestellte Anwendung ist in JavaScript mit der Node.js-Laufzeit geschrieben. Es handelt sich um eine "hello, world"-Anwendung, die auf dem Express-Framework basiert. Erstellen Sie die Anwendung entsprechend den folgenden Anforderungen, und stellen Sie sie auf einem OpenShift-Cluster bereit:

- Anwendungscode wird aus einem neuen Branch mit dem Namen `source-build` bereitgestellt.

Kapitel1 | Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster

- Der Projektnname für OpenShift lautet `youruser-source-build`.
- Der Anwendungsname für OpenShift lautet `greet`.
- Der Zugriff auf die Anwendung sollte über die Standardroute erfolgen:
`greet-youruser-source-build.apps.cluster.domain.example.com`
- Das Git-Repository, das die Anwendungsverzeichnisquellen enthält, lautet:
`https://github.com/yourgithubuser/D0288-apps/nodejs-helloworld`.
- Die zum Entwickeln der Anwendung erforderlichen NPM-Module sind verfügbar unter:
`http://nexus-common.apps.cluster.domain.example.com/repository/nodejs`
Verwenden Sie die Build-Umgebungsvariable `npm_config_registry`, um diese Informationen an das S2I-Builder-Image für Node.js weiterzugeben.
- Sie können den Befehl `python -m json.tool filename.json` verwenden, um Syntaxfehler in JSON-Dateien zu ermitteln.

Anweisungen

1. Navigieren Sie zu Ihrem lokalen Klon des D0288-apps-Git-Repository, und erstellen Sie aus dem Main-Branch einen neuen Branch mit dem Namen `source-build`. Stellen Sie die Anwendung im Ordner `nodejs-helloworld` für das Projekt `youruser-source-build` auf dem OpenShift-Cluster bereit.
2. Zeigen Sie die Build-Logs an, um den Build-Fehler zu ermitteln, und überprüfen Sie die Anwendungsquellen, um die Ursache zu ermitteln.
Denken Sie daran, dass Sie mit dem Befehl `python3 -m json.tool filename` eine JSON-Datei überprüfen können.
3. Korrigieren Sie den Fehler in der Konfigurationsdatei des Build-Tools, und übertragen Sie die Änderungen an das Git-Repository.
4. Starten Sie einen neuen Build der Anwendung und überprüfen Sie, ob die Anwendung erfolgreich bereitgestellt wird.
5. Vergewissern Sie sich, dass die AnwendungsLogs keine Fehler aufweisen, stellen Sie die Anwendung für den externen Zugriff bereit und überprüfen Sie, ob die Anwendung die Meldung „hello, world“ zurückgibt.

Bewertung

Verwenden Sie als Benutzer `student` auf dem Rechner `workstation` den Befehl `lab`, um Ihre Arbeit zu bewerten. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie den Befehl so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab source-build grade
```

Beenden

Führen Sie auf der workstation den Befehl `lab source-build finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken.

```
[student@workstation ~]$ lab source-build finish
```

Hiermit ist die praktische Übung abgeschlossen.

► Lösung

Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster

In dieser praktischen Übung stellen Sie eine Anwendung für ein OpenShift-Cluster anhand des Quellcodes bereit. Die Build-Konfigurationsdatei der Anwendung weist einen Fehler auf, den Sie diagnostizieren und beheben sollen.



Anmerkung

Für den Befehl `grade` am Ende jeder praktischen Übung eines Kapitels ist es erforderlich, dass Sie die exakten Projektnamen und anderen Identifikatoren, wie in der Spezifikation der praktischen Übung angegeben, verwenden.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Erstellen einer Anwendung mit der Quellcode-Build-Strategie
- Überprüfen der Build-Logs zum Suchen von Informationen zu einem Build-Fehler
- Aktualisieren der Build-Tool-Konfiguration der Anwendung zum Beheben des Build-Fehlers
- Erneutes Erstellen der Anwendung und Verifizieren, ob sie erfolgreich bereitgestellt wurde

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen aktiven OpenShift-Cluster
- Auf das S2I-Builder-Image für Node.js 12-Anwendungen
- Auf die Anwendung im Git-Repository (`nodejs-helloworld`)

Führen Sie den folgenden Befehl auf `workstation` aus, um die Voraussetzungen zu überprüfen. Der Befehl lädt auch die Hilfs- und Lösungsdateien für die Wiederholungsübung herunter:

```
[student@workstation ~]$ lab source-build start
```

Anforderungen

Die bereitgestellte Anwendung ist in JavaScript mit der Node.js-Laufzeit geschrieben. Es handelt sich um eine "hello, world"-Anwendung, die auf dem Express-Framework basiert. Erstellen Sie die Anwendung entsprechend den folgenden Anforderungen, und stellen Sie sie auf einem OpenShift-Cluster bereit:

- Anwendungscode wird aus einem neuen Branch mit dem Namen `source-build` bereitgestellt.

- Der Projektname für OpenShift lautet *youruser-source-build*.
 - Der Anwendungsname für OpenShift lautet *greet*.
 - Der Zugriff auf die Anwendung sollte über die Standardroute erfolgen:
`greet-youruser-source-build.apps.cluster.domain.example.com`
 - Das Git-Repository, das die Anwendungsverzeichnisquellen enthält, lautet:
`https://github.com/yourgithubuser/D0288-apps/nodejs-helloworld`.
 - Die zum Entwickeln der Anwendung erforderlichen NPM-Module sind verfügbar unter:
`http://nexus-common.apps.cluster.domain.example.com/repository/nodejs`
- Verwenden Sie die Build-Umgebungsvariable `npm_config_registry`, um diese Informationen an das S2I-Builder-Image für Node.js weiterzugeben.
- Sie können den Befehl `python -m json.tool filename.json` verwenden, um Syntaxfehler in JSON-Dateien zu ermitteln.

Anweisungen

1. Navigieren Sie zu Ihrem lokalen Klon des D0288-apps-Git-Repository, und erstellen Sie aus dem Main-Branch einen neuen Branch mit dem Namen `source-build`. Stellen Sie die Anwendung im Ordner `nodejs-helloworld` für das Projekt `youruser-source-build` auf dem OpenShift-Cluster bereit.

- 1.1. Bereiten Sie die Umgebung für die praktische Übung vor.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

- 1.2. Melden Sie sich bei OpenShift an und erstellen Sie das Projekt:

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful
...output omitted...
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-source-build
...output omitted...
```

- 1.3. Wechseln Sie zu Ihrem lokalen Klon des D0288-apps-Git-Repository und checken Sie den `main`-Branch des Kurs-Repository aus, um sicherzustellen, dass Sie diese Übung von einem als fehlerfrei bekannten Zustand aus starten:

```
[student@workstation ~]$ cd D0288-apps
[student@workstation D0288-apps]$ git checkout main
...output omitted...
```

- 1.4. Erstellen Sie einen neuen Branch, in dem Sie alle Änderungen speichern können, die Sie während dieser Übung vornehmen:

```
[student@workstation D0288-apps]$ git checkout -b source-build
Switched to a new branch 'source-build'
[student@workstation D0288-apps]$ git push -u origin source-build
...output omitted...
* [new branch]      source-build -> source-build
Branch source-build set up to track remote branch source-build from origin.
```

- 1.5. Erstellen Sie eine neue Anwendung aus den Quellen im Git-Repository. Nennen Sie die Anwendung `greet`. Verwenden Sie die Option `--build-env` mit dem Befehl `oc new-app`, um die Build-Umgebungsvariable mit der URL für die NPM-Module zu definieren.

Kopieren Sie entweder den Befehl aus dem Skript `oc-new-app.sh` im Ordner `/home/student/D0288/labs/source-build` oder führen Sie den Befehl aus:

```
[student@workstation D0288-apps]$ oc new-app --name greet \
--build-env npm_config_registry=\
http://${RHT_OCP4_NEXUS_SERVER}/repository/nodejs \
nodejs:12-https://github.com/${RHT_OCP4_GITHUB_USER}/D0288-apps#source-build \
--context-dir nodejs-helloworld
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "greet" created
buildconfig.build.openshift.io "greet" created
deployment.apps "greet" created
service "greet" created
--> Success
...output omitted...
```



Anmerkung

Vor und hinter dem Gleichheitszeichen (=) steht nach `npm_config_registry` kein Leerzeichen. Das komplette `Schlüssel=Wert`-Paar für die Build-Umgebungsvariable ist zu lang für die Seitenbreite.

2. Zeigen Sie die Build-Logs an, um den Build-Fehler zu ermitteln, und überprüfen Sie die Anwendungsquellen, um die Ursache zu ermitteln.

Denken Sie daran, dass Sie mit dem Befehl `python3 -m json.tool filename` eine JSON-Datei überprüfen können.

- 2.1. Beobachten Sie die Build-Logs:

```
[student@workstation D0288-apps]$ oc logs -f bc/greet
```

Sie sollten eine JSON-Analysefehlermeldung sehen:

```
...output omitted...
--> Installing all dependencies
npm ERR! code EJSONPARSE
npm ERR! file /opt/app-root/src/package.json
npm ERR! JSON.parse Failed to parse json
npm ERR! JSON.parse Unexpected string in JSON at position 241 while parsing '{
```

```
npm ERR! JSON.parse    "name": "nodejs-helloworld",
npm ERR! JSON.parse    "vers"
npm ERR! JSON.parse Failed to parse package.json data.
npm ERR! JSON.parse package.json must be actual JSON, not just JavaScript.
...output omitted...
```

Das Node.js-Builder-Image enthält keine spezifische Fehlerposition in der Konfigurationsdatei `package.json` des Build-Tools.

- 2.2. Führen Sie den Befehl `python3 -m json.tool` aus, um die JSON-Datei zu überprüfen:

```
[student@workstation D0288-apps]$ python3 -m json.tool \
nodejs-helloworld/package.json
```

Es sollte die folgende Fehlermeldung angezeigt werden:

```
Expecting : delimiter: line 12 column 15 (char 241)
```

- 2.3. Öffnen Sie die Konfigurationsdatei `~/D0288-apps/nodejs-helloworld/package.json` des Build-Tools in einem Texteditor und ermitteln Sie den Syntaxfehler. Die folgende teilweisen Auflistung zeigt, dass hinter dem Schlüssel `express` ein Doppelpunkt (`:`) fehlt:

```
"dependencies": {
  "express" "4.14.x"
}
```

3. Korrigieren Sie den Fehler in der Konfigurationsdatei des Build-Tools, und übertragen Sie die Änderungen an das Git-Repository.

- 3.1. Bearbeiten Sie die Quelldatei `~/D0288-apps/nodejs-helloworld/package.json`, um nach dem Schlüssel `express` einen Doppelpunkt (`:`) einzufügen.

Die endgültigen Dateiinhalte sollten wie folgt aussehen:

```
"dependencies": {
  "express": "4.14.x"
}
```

- 3.2. Übergeben und übertragen Sie die Korrekturen an das Git-Repository:

```
[student@workstation D0288-apps]$ cd nodejs-helloworld
[student@workstation nodejs-helloworld]$ git commit -a -m 'Fixed JSON syntax'
...output omitted...
[student@workstation nodejs-helloworld]$ git push
...output omitted...
[student@workstation nodejs-helloworld]$ cd ~
[student@workstation ~]$
```

4. Starten Sie einen neuen Build der Anwendung und überprüfen Sie, ob die Anwendung erfolgreich bereitgestellt wird.

Kapitel1 | Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster

- 4.1. Starten Sie einen neuen Build der Anwendung greet und beobachten Sie deren Logs. Warten Sie, bis der Build ohne Fehler abgeschlossen wird:

```
[student@workstation ~]$ oc start-build --follow bc/greet
build "greet-2" started
...output omitted...
Push successful
```

- 4.2. Verifizieren Sie, dass eine neue Bereitstellung gestartet wird:

```
[student@workstation ~]$ oc status
...output omitted...
svc/greet - 172.30.160.185:8080
  deployment/greet deploys istag/greet:latest <-
    bc/greet source builds https://github.com/yourgithubuser/D0288-apps#source-
  build on openshift/nodejs:10
    deployment #2 running for 23 seconds - 1 pod
...output omitted...
```

- 4.3. Warten Sie, bis der Anwendungs-Pod bereit ist und ausgeführt wird:

```
[student@workstation ~]$ oc get pod
NAME          READY   STATUS    RESTARTS   AGE
greet-1-build  0/1     Error     0          4m
greet-2-build  0/1     Completed  0          2m
greet-594d667bc4-2b9ch 1/1     Running   0          59s
```

5. Vergewissern Sie sich, dass die AnwendungsLogs keine Fehler aufweisen, stellen Sie die Anwendung für den externen Zugriff bereit und überprüfen Sie, ob die Anwendung die Meldung „hello, world“ zurückgibt.

- 5.1. Greifen Sie auf die AnwendungsLogs zu. Es sollten keine Fehlermeldungen von der Anwendung angezeigt werden.

```
[student@workstation ~]$ oc logs greet-1-gf59d
...output omitted...
Example app listening on port 8080!
```

- 5.2. Stellen Sie die Anwendung für den externen Zugriff bereit:

```
[student@workstation ~]$ oc expose svc/greet
route.route.openshift.io/greet exposed
```

- 5.3. Rufen Sie den Hostnamen ab, der von OpenShift für die neue Route generiert wurde:

```
[student@workstation ~]$ oc get route
NAME      HOST/PORT
greet     greet-youruser-source-build.apps.cluster.domain.example.com ...
```

- 5.4. Senden Sie mit dem Befehl curl und dem Hostnamen aus dem vorherigen Schritt eine HTTP-Anforderung an die Anwendung. Es wird die Meldung „hello, world“ zurückgegeben:

```
[student@workstation ~]$ curl \
http://greet-${RHT_OCP4_DEV_USER}-source-build.${RHT_OCP4_WILDCARD_DOMAIN}
Hello, World!
```

Bewertung

Verwenden Sie als Benutzer `student` auf dem Rechner `workstation` den Befehl `lab`, um Ihre Arbeit zu bewerten. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie den Befehl so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab source-build grade
```

Beenden

Führen Sie auf der `workstation` den Befehl `lab source-build finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken.

```
[student@workstation ~]$ lab source-build finish
```

Hiermit ist die praktische Übung abgeschlossen.

Zusammenfassung

In diesem Kapitel wurden die folgenden Themen behandelt:

- RHOPC stellt ein PaaS-Tool bereit, das auf Red Hat CoreOS und Kubernetes ausgeführt wird.
- OpenShift unterstützt das Erstellen von Container-Images aus Anwendungsquellcode oder anhand des S2I-Prozesses direkt über Dockerfiles.
- Ressourcen für Build- und Deploymentkonfigurationen automatisieren die Build- und Deploymentprozesse und können automatisch auf Änderungen im Anwendungsquellcode oder auf Aktualisierungen reagieren, die an Container-Images vorgenommen werden.
- Der Befehl `oc new-app` kann die Quellprogrammiersprache automatisch erkennen, die von einer Anwendung in einem Git-Repository verwendet wurde. Darüber hinaus bietet er eine Reihe von Optionen zur Auflösen von Mehrdeutigkeiten.
- Nützliche `oc`-Sub-Befehle für die Fehlerbehebung von Builds und Bereitstellungen sind `get` , `describe`, `edit`, `logs`, `cp` und `rsh`.
- Entwickler verfügen möglicherweise nicht über Cluster-Administratorberechtigungen für ihre Entwicklungsumgebung oder sie haben nur für eine Teilmenge aller Projekte im OpenShift-Cluster Projektadministrator- oder Bearbeitungsberechtigungen.

Kapitel 2

Entwerfen containerisierter Anwendungen für OpenShift

Ziel

Auswählen einer Methode für die Containerisierung einer Anwendung und Packen der Anwendung zur Ausführung auf einem OpenShift-Cluster

Ziele

- Auswählen einer geeigneten Methode für die Containerisierung von Anwendungen
- Erstellen eines Container-Images mit zusätzlichen Anweisungen für Dockerfiles
- Auswählen einer Methode zum Einfügen von Konfigurationsdaten in eine Anwendung und Erstellen der dazu erforderlichen Ressourcen

Abschnitte

- Auswählen eines Ansatzes für die Containerisierung (und Test)
- Erstellen von Container-Images mit zusätzlichen Anweisungen für Containerfiles (und angeleitete Übung)
- Einfügen von Konfigurationsdaten in eine Anwendung (und angeleitete Übung)

Praktische Übung

Entwerfen containerisierter Anwendungen für OpenShift

Auswählen eines Ansatzes für die Containerisierung

Ziele

Nach Abschluss dieses Abschnitts sollten Sie in der Lage sein, eine geeignete Methode für die Anwendungscontainerisierung auszuwählen.

Auswählen einer Build-Methode

Die grundlegende Deploymenteinheit in OpenShift ist ein Container-Image, auch einfach als Image bezeichnet. Ein Container-Image besteht aus der Anwendung und allen ihren Abhängigkeiten wie freigegebene Bibliotheken, Laufzeitumgebungen, Interpreter und mehr. Es gibt verschiedene Möglichkeiten, Container-Images zu erstellen. Dies hängt vom Anwendungstyp ab, den Sie auf einem OpenShift-Cluster bereitstellen und ausführen möchten:

Container-Images

Außerhalb von OpenShift erstellte Container-Images können direkt in einem OpenShift-Cluster bereitgestellt werden. Dieser Ansatz eignet sich in Fällen, in denen Sie bereits über eine als Container-Image gepackte Anwendung verfügen. Sie können diesen Ansatz auch für ein von einem Drittanbieter bereitgestelltes, zertifiziertes und unterstütztes Container-Image verwenden. Sie können von einem Drittanbieter erstellte Images in einem OpenShift-Cluster bereitstellen.

Dockerfiles

In bestimmten Fällen wird Ihnen das zum Erstellen des Anwendungscontainer-Images verwendete Dockerfile bereitgestellt. In solchen Szenarien gibt es weitere, zu berücksichtigende Optionen:

- Sie können das Dockerfile anpassen und neue Images erstellen, um den Anforderungen Ihrer Anwendung gerecht zu werden. Verwenden Sie diese Option, wenn die Änderungen geringfügig sind und Sie dem Image nicht zu viele Layer hinzufügen möchten.
- Sie können ein Dockerfile mit dem bereitgestellten Container-Image als übergeordnetes Element erstellen und das Basis-Image an die Anforderungen Ihrer Anwendung anpassen. Verwenden Sie diese Option, wenn Sie ein neues untergeordnetes Image mit weiteren Anpassungen erstellen möchten, das die Layer des übergeordneten Images übernimmt.

S2I-Builder-Images (Source-to-Image)

Ein S2I-Builder-Image umfasst grundlegende Betriebssystem-Libraries, Compiler, Interpreter, Laufzeiten, Frameworks und Source-to-Image-Tooling. Wenn eine Anwendung mit diesem Ansatz entwickelt wird, kombiniert OpenShift den Anwendungsquellcode mit dem Builder-Image, um ein einsatzbereites Container-Image zu erstellen, das OpenShift dann für einen Cluster bereitstellen kann. Dieser Ansatz bietet verschiedene Vorteile für Entwickler und stellt die schnellste Möglichkeit zum Entwickeln neuer Anwendungen für die Bereitstellung in einem OpenShift-Cluster dar.

In Abhängigkeit der Anforderungen Ihrer Anwendung stehen Ihnen verschiedene Möglichkeiten zum Verwenden von S2I-Builder-Images zur Verfügung:

- Red Hat bietet viele unterstützte S2I-Builder-Images für verschiedene Anwendungstypen. Red Hat empfiehlt, dass Sie nach Möglichkeit eines dieser standardmäßigen S2I-Builder-Images verwenden.

- S2I-Builder-Images entsprechen regulären Container-Images, weisen jedoch zusätzliche Metadaten, Skripts und Tooling im Image auf. Sie können Dockerfiles zum Erstellen von untergeordneten Images auf Grundlage der übergeordneten Builder-Images verwenden, die von Red Hat bereitgestellt werden.
- Wenn keines der standardmäßigen, von Red Hat bereitgestellten S2I-Builder-Images den Anforderungen Ihrer Anwendung gerecht wird, können Sie ein eigenes benutzerdefiniertes S2I-Builder-Image erstellen.

S2I-Builder-Images

Ein S2I-Builder-Image ist eine spezialisierte Form eines Container-Images, das als Ergebnis Anwendungscontainer-Images erzeugt. Builder-Images enthalten grundlegende Betriebssystem-Libraries, Sprachlaufzeiten, Frameworks und Libraries, von denen eine Anwendung abhängig ist, sowie Source-to-Images-Tools und -Dienstprogramme.

Wenn Sie beispielsweise eine in PHP geschriebene Anwendung in OpenShift bereitstellen möchten, können Sie ein PHP-Builder-Image verwenden, um ein Anwendungscontainer-Image zu erzeugen. Sie geben den Speicherort des Git-Repository an, in dem der Anwendungsquellcode gespeichert ist, und OpenShift kombiniert den Quellcode und das grundlegende Builder-Image, um ein Container-Image zu erzeugen, das OpenShift in dem Cluster bereitstellen kann. Das resultierende Anwendungscontainer-Image enthält eine Version von Red Hat Enterprise Linux, eine PHP-Laufzeit und die Anwendung. Builder-Images sind praktische Mechanismen, um schnell und einfach aus Code einen ausführbaren Container zu erhalten, ohne Dockerfiles erstellen zu müssen.

Verwenden von Container-Images

Obwohl Source-to-Image-Builds die bevorzugte Möglichkeit zum Entwickeln und Bereitstellen von Anwendungen in OpenShift darstellen, gibt es möglicherweise Szenarien, in denen Sie von einem Drittanbieter entwickelte Anwendungen bereitstellen müssen. Beispielsweise stellen bestimmte Anbieter vollständig zertifizierte und unterstützte Container-Images bereit, die einsatzbereit sind. In solchen Fällen unterstützt OpenShift Bereitstellungen der vorab erstellten Container-Images.

Der Befehl `oc new-app` bietet verschiedene flexible Möglichkeiten, Container-Images in einem OpenShift-Cluster bereitzustellen. Der einfachste Ansatz besteht darin, das vorab erstellte Image über eine öffentliche (wie `docker.io` oder `quay.io`) oder private (in Ihrer Organisation gehostete) Registry zur Verfügung zu stellen und dann im Befehl `oc new-app` den Speicherort dieses Images anzugeben. OpenShift ruft das Image anschließend ab und stellt es wie jedes andere in OpenShift erstellte Image in einem OpenShift-Cluster bereit.



Anmerkung

Ein Beispieldokument, das die Bereitstellungen eines vorab erstellten Container-Images in einem OpenShift-Cluster veranschaulicht, finden Sie unter OpenShift Binary Deployments [<https://blog.openshift.com/binary-deployments-openshift-3/>].

Verwenden von Red Hat Container Catalog

Der *Red Hat Container Catalog* ist eine vertrauenswürdige Quelle sicherer, zertifizierter und aktueller Container-Images. Er enthält einfache Container-Images und S2I-Builder-Images. Für unternehmenskritische Anwendungen sind vertrauenswürdige Container erforderlich. Red Hat erstellt die Container Catalog-Container-Images aus RPM-Ressourcen, die vom internen Red Hat-Sicherheitsteam untersucht und gegen Sicherheitslücken geschützt wurden.

Kapitel 2 | Entwerfen containerisierter Anwendungen für OpenShift

Das Red Hat Container Catalog-Portal unter <https://registry.redhat.io> enthält Informationen hinsichtlich einiger Container-Images, die Red Hat auf Versionen von Red Hat Enterprise Linux (RHEL) und zugehörigen Systemen erstellt. Es enthält einige einsatzbereite Container-Images, die zum Bereitstellen von Anwendungen für OpenShift verwendet werden können.

Red Hat verwendet einen *Container Health Index* für die Sicherheitsrisikobewertung der Container-Images, die Red Hat im Red Hat Container Catalog bereitstellt. Weitere Informationen über das von Red Hat im Container Health Index verwendete Einstufungssystem finden Sie unter <https://access.redhat.com/articles/2803031>.

Weitere Details zum Red Hat Container Catalog finden Sie unter Frequently Asked Questions (FAQ) [<https://access.redhat.com/containers/#/faq>].

Auswählen von Container-Images für Anwendungen

Die Auswahl eines Container-Images für Ihre Anwendung hängt von einigen Faktoren ab. Wenn Sie einen benutzerdefinierten Container erstellen möchten, sollten Sie mit einem grundlegenden Betriebssystem-Image (wie `rhel7`) beginnen. Zum Entwickeln und Ausführen von Anwendungen, für die bestimmte Entwicklungstools und Laufzeit-Libraries erforderlich sind, stellt Red Hat Container-Images bereit, die Tools wie Node.js (`rhscl/nodejs-8-rhel7`), Ruby on Rails (`rhscl/ror-50-rhel7`) und Python (`rhscl/python-36-rhel7`) umfassen.

Die *Red Hat Software Collections Library (RHSCl)*, oder einfach Software Collections, ist die Lösung von Red Hat für Entwickler, die die neuesten Entwicklungstools verwenden müssen, die jedoch in der Regel nicht dem Standardversionsplan von Red Hat Enterprise Linux (RHEL) entsprechen. Red Hat bietet im Red Hat Container Catalog viele Container-Images als Bestandteil der RHSCl.

Red Hat stellt auch Red Hat OpenShift Application Runtimes (RHOAR), die Entwicklungsplattform von Red Hat für Cloud-native und Microservices-Anwendungen, bereit. RHOAR bietet einen von Red Hat optimierten und unterstützten Ansatz für die Entwicklung von Microservices-Anwendungen für OpenShift als Deploymentplattform.

RHOAR unterstützt mehrere Laufzeiten, Sprachen, Frameworks und Architekturen. Es ermöglicht die Auswahl und die Flexibilität, die richtigen Frameworks und Laufzeiten für den richtigen Job auszuwählen. Anwendungen, die mit RHOAR entwickelt wurden, können auf jeder Infrastruktur ausgeführt werden, in der Red Hat OpenShift Container Platform ausgeführt werden kann, und ermöglichen so die Umgehung von Vendor Lock-ins.

RHOAR bietet:

- Zugriff auf von Red Hat erstellte und unterstützte Binärdateien für ausgewählte Microservices-Entwicklungs-Frameworks und Laufzeiten
- Zugriff auf von Red Hat erstellte und unterstützte Binärdateien für Integrationsmodule, die zur Verwendung von OpenShift-Funktionen die Implementierung eines Microservices-Patterns eines Frameworks ersetzen oder verbessern
- Entwicklerunterstützung für das Schreiben von Anwendungen mithilfe ausgewählter Microservice-Entwicklungs-Frameworks, Laufzeiten, Integrationsmodule und Integration mit ausgewählten externen Services, z. B. Datenbankservern
- Produktionsunterstützung für die Bereitstellung von Anwendungen mithilfe ausgewählter Microservice-Entwicklungs-Frameworks, Laufzeiten, Integrationsmodule und Integrationen in einem unterstützten OpenShift-Cluster

Erstellen von S2I-Builder-Images

Wenn Ihre Anwendungen ein benutzerdefiniertes S2I-Builder-Image mit Ihrem benutzerdefinierten Satz von Laufzeiten, Skripts, Frameworks und Libraries verwenden sollen, können Sie ein eigenes S2I-Builder-Image erstellen. Für das Erstellen von S2I-Builder-Images gibt es mehrere Möglichkeiten:

- Erstellen eines neuen eigenen S2I-Builder-Images. In Szenarien, in denen Ihre Anwendung die im Container Catalog bereitgestellten S2I-Builder-Images nicht unverändert verwenden kann, können Sie ein benutzerdefiniertes S2I-Builder-Image erstellen, um den Build-Prozess an die Anforderungen Ihrer Anwendung anzupassen.

OpenShift stellt das Befehlszeilentool `s2i` bereit, mit dem Sie ein Bootstrapping der Build-Umgebung für das Erstellen benutzerdefinierter S2I-Builder-Images vornehmen können. Es ist im Paket `source-to-image` in den Yum-Repositorys der RHSCl (`rhel-server-rhscl-7-rpms`) verfügbar.

- Forken eines vorhandenen S2I-Builder-Images. Anstatt von Grund auf neu zu beginnen, können Sie die Dockerfiles für die vorhandenen Builder-Images im Container Catalog verwenden, die unter <https://github.com/sclorg/?q=s2i> verfügbar sind, und sie an Ihre Anforderungen anpassen.
- Erweitern eines vorhandenen S2I-Builder-Images. Sie können auch ein vorhandenes Builder-Image erweitern, indem Sie ein untergeordnetes Image erstellen und dann Inhalt aus dem vorhandenen Builder-Image hinzufügen oder ersetzen.

Ein gutes Tutorial, in dem Sie durch das Erstellen eines benutzerdefinierten S2I-Builder-Images geführt werden, finden Sie unter <https://blog.openshift.com/create-s2i-builder-image/>



Literaturhinweise

Red Hat Container Catalog

<https://access.redhat.com/containers>

Dockerfiles für Images, die Bestandteil der Red Hat Software Collections Library sind, finden Sie unter
<https://github.com/sclorg?q=-container>

Weitere Informationen zu Container-Images, die von Red Hat für die Verwendung mit OpenShift unterstützt werden, finden Sie im Kapitel *Creating Images* des Handbuchs *Images* für Red Hat OpenShift Container Platform 4.6 unter
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/images/index#creating-images

► Quiz

Auswählen eines Ansatzes für die Containerisierung

Wählen Sie die richtigen Antworten auf die folgenden Fragen aus:

Klicken Sie nach Abschluss des Tests auf **CHECK**. Wenn Sie es noch einmal versuchen möchten, klicken Sie auf **RESET**. Klicken Sie auf **SHOW SOLUTION**, damit alle korrekten Antworten angezeigt werden.

- 1. **Sie wurden gebeten, eine kommerzielle,.NET-basierte Drittanbieteranwendung in einem OpenShift-Cluster bereitzustellen, die vom Anbieter als Container-Image gepackt wurde. Welche der folgenden Optionen würden Sie zum Bereitstellen der Anwendung verwenden?**
- a. Ein Source-to-Image-Build.
 - b. Einen benutzerdefinierten Source-to-Image-Builder.
 - c. Bereitstellen des Container-Images in einer privaten Container-Registry mit anschließender Bereitstellung des Container-Images in einem OpenShift-Cluster mit dem Befehlszeilentool oc von OpenShift.
 - d. Nichts davon. Es ist nicht möglich,auf .NET basierte Anwendungen in einem OpenShift-Cluster bereitzustellen.

► **2. Sie werden beauftragt, eine RHEL 7-basierte, benutzerdefinierte, intern entwickelte C++-Anwendung in einem OpenShift-Cluster bereitzustellen. Sie erhalten den vollständigen Quellcode der Anwendung. Welche der folgenden Optionen können entsprechend den Best Practices zum Entwickeln eines Container-Images verwendet werden, um es in einem OpenShift-Cluster bereitzustellen? (Wählen Sie zwei Antworten aus.)**

- a. Erstellen eines Dockerfiles mit dem Container-Image `rhel7` aus dem Red Hat Container Catalog als Grundlage für die Anwendung und Bereitstellen dieser in OpenShift mit einem Git-Repository. OpenShift kann aus dem bereitgestellten Dockerfile ein Container-Image erstellen.
- b. Herunterladen eines auf CentOS 7/C++ basierten Container-Images aus einer öffentlichen Registry wie `docker.io` und Erstellen eines Dockerfiles, das die Anwendung kompiliert und packt. Da CentOS 7-basierte Binärdateien mit RHEL 7 binärkompatibel sind, funktioniert die Anwendung ordnungsgemäß, wenn sie in einem OpenShift-Cluster bereitgestellt wird.
- c. Erstellen eines Dockerfiles mit dem RHEL 7-Container-Image aus dem Red Hat Container Catalog als Grundlage und anschließendes Erstellen eines Container-Images auf Grundlage des Dockerfiles. Bereitstellen des Container-Images in einem OpenShift-Cluster mit dem Befehlszeilentool `oc` von OpenShift.
- d. Erstellen eines Dockerfiles mit dem Linux-basierten Container-Image aus einer öffentlichen Registry als Grundlage und anschließendes Erstellen eines Container-Images auf Grundlage des Dockerfiles. Bereitstellen des Container-Images in einem OpenShift-Cluster mit dem Befehlszeilentool `oc` von OpenShift.

- **3. Sie wurden beauftragt, auf Basis von Ruby on Rails bzw. Node.js zwei Anwendungen zum OpenShift-Cluster zu migrieren bzw. in diesem bereitzustellen. Diese Anwendungen wurden zuvor in einer Umgebung mit virtuellen Rechnern ausgeführt. Ihnen wurde der Speicherort der Git-Repositorys mitgeteilt, an dem sich der Anwendungsquellcode für beide Anwendungen befindet. Welche der folgenden Optionen wird zum Bereitstellen der Anwendungen in einem OpenShift-Cluster empfohlen? Beachten Sie dabei, dass Sie beauftragt wurden, diese Anwendungen um weitere Funktionen zu erweitern und die Entwicklung in einer OpenShift-Umgebung fortzusetzen.**
- a. Verwenden der Ruby on Rails- und Node.js-Container-Images von docker.io als Grundlage. Erstellen benutzerdefinierter Dockerfiles für die beiden Anwendungen und Erstellen von Container-Images aus ihnen. Bereitstellen der Images in einem OpenShift-Cluster mit dem StandardDeploymentprozess für Binärdateien.
 - b. Erstellen von benutzerdefinierten S2I-basierten Builder-Images, da weder für Ruby on Rails noch für Node.js in OpenShift Builder-Images verfügbar sind.
 - c. Verwenden der Ruby on Rails- und Node.js-S2I-Builder-Images aus dem Red Hat Container Catalog und Bereitstellen der Anwendungen in einem OpenShift-Cluster mit einem S2I-Build-Standardprozess.
 - d. Verwenden der einfachen Ruby- und Node.js-basierten Container-Images aus dem Red Hat Container Catalog und Erstellen von benutzerdefinierten Dockerfiles für jedes von ihnen. Erstellen und Bereitstellen der resultierenden Container-Images in einem OpenShift-Cluster.

- **4. Sie wurden beauftragt, eine in der Go-Programmiersprache geschriebene Webanwendung in einem OpenShift-Cluster bereitzustellen. Das Sicherheitsteam Ihrer Organisation möchte, dass alle Anwendungen über ein statisches Quellcode-Analysesystem und über eine Suite von automatisierten Modul- und Integrationstests ausgeführt werden, bevor sie in Produktionsumgebungen bereitgestellt werden. Das Sicherheitsteam hat zudem ein benutzerdefiniertes Dockerfile bereitgestellt. Dieses sorgt dafür, dass alle Anwendungen auf einem RHEL 7-basierten Betriebssystem basierend auf dem RHEL 7-Standard-Image aus dem Red Hat Container Catalog bereitgestellt werden. Die Umgebung enthält eine verwaltete Liste von Paketen, Benutzern und angepassten Konfigurationen für die Kernservices im Betriebssystem. Darüber hinaus hat der Anwendungsarchitekt darauf bestanden, dass es eine klare Trennung zwischen Änderungen auf Quellcodeebene und Infrastrukturänderungen (Betriebssystem, Go-Compiler und Go-Tools) gibt. Änderungen am und Patches für das Betriebssystem oder die Go-Runtime-Layer sollten automatisch eine Neuerstellung und Neubereitstellung der Anwendung auslösen. Welche der folgenden Optionen würden Sie zum Erreichen dieses Ziels verwenden?**
- a. Erstellen eines benutzerdefinierten Dockerfiles, das ein Anwendungscontainer-Image erstellt, das aus der RHEL 7-Betriebssystembasis, der Go-Laufzeit und dem Analysetool besteht. Bereitstellen des resultierenden Images in einem OpenShift-Cluster mit dem Deploymentprozess für Binärdateien.
 - b. Erstellen separater Dockerfiles für die RHEL 7-Betriebssystembasis, für die Go-Laufzeit und für das Analysetool. OpenShift kann die Dockerfiles automatisch zusammenführen, um ein einzelnes, ausführbares Anwendungscontainer-Image zu erzeugen.
 - c. Erstellen eines benutzerdefinierten S2I-Builder-Images für diese Anwendung, die das statische Analysetool, den Go-Compiler und die Laufzeit sowie das RHEL 7-Betriebssystem-Image auf Grundlage des Dockerfiles einbettet.
 - d. Erstellen separater Container-Images für die RHEL 7-Betriebssystembasis, für die Go-Laufzeit und für das Analysetool. Nachdem diese Images in einer privaten oder öffentlichen Container-Image-Registry bereitgestellt wurden, kann OpenShift die Layer aus den einzelnen Images automatisch verketten, um das endgültige ausführbare Anwendungscontainer-Image zu erstellen.
 - e. Nichts davon. Diese Anforderung kann nicht erfüllt werden und die Anwendung kann nicht in einem OpenShift-Cluster bereitgestellt werden.

► Lösung

Auswählen eines Ansatzes für die Containerisierung

Wählen Sie die richtigen Antworten auf die folgenden Fragen aus:

Klicken Sie nach Abschluss des Tests auf **CHECK**. Wenn Sie es noch einmal versuchen möchten, klicken Sie auf **RESET**. Klicken Sie auf **SHOW SOLUTION**, damit alle korrekten Antworten angezeigt werden.

- 1. **Sie wurden gebeten, eine kommerzielle,.NET-basierte Drittanbieteranwendung in einem OpenShift-Cluster bereitzustellen, die vom Anbieter als Container-Image gepackt wurde. Welche der folgenden Optionen würden Sie zum Bereitstellen der Anwendung verwenden?**
- a. Ein Source-to-Image-Build.
 - b. Einen benutzerdefinierten Source-to-Image-Builder.
 - c. Bereitstellen des Container-Images in einer privaten Container-Registry mit anschließender Bereitstellung des Container-Images in einem OpenShift-Cluster mit dem Befehlszeilentool `oc` von OpenShift.
 - d. Nichts davon. Es ist nicht möglich, auf .NET basierte Anwendungen in einem OpenShift-Cluster bereitzustellen.

► **2. Sie werden beauftragt, eine RHEL 7-basierte, benutzerdefinierte, intern entwickelte C++-Anwendung in einem OpenShift-Cluster bereitzustellen. Sie erhalten den vollständigen Quellcode der Anwendung. Welche der folgenden Optionen können entsprechend den Best Practices zum Entwickeln eines Container-Images verwendet werden, um es in einem OpenShift-Cluster bereitzustellen? (Wählen Sie zwei Antworten aus.)**

- a. Erstellen eines Dockerfiles mit dem Container-Image `rhel7` aus dem Red Hat Container Catalog als Grundlage für die Anwendung und Bereitstellen dieser in OpenShift mit einem Git-Repository. OpenShift kann aus dem bereitgestellten Dockerfile ein Container-Image erstellen.
- b. Herunterladen eines auf CentOS 7/C++ basierten Container-Images aus einer öffentlichen Registry wie `docker.io` und Erstellen eines Dockerfiles, das die Anwendung kompiliert und packt. Da CentOS 7-basierte Binärdateien mit RHEL 7 binärkompatibel sind, funktioniert die Anwendung ordnungsgemäß, wenn sie in einem OpenShift-Cluster bereitgestellt wird.
- c. Erstellen eines Dockerfiles mit dem RHEL 7-Container-Image aus dem Red Hat Container Catalog als Grundlage und anschließendes Erstellen eines Container-Images auf Grundlage des Dockerfiles. Bereitstellen des Container-Images in einem OpenShift-Cluster mit dem Befehlszeilentool `oc` von OpenShift.
- d. Erstellen eines Dockerfiles mit dem Linux-basierten Container-Image aus einer öffentlichen Registry als Grundlage und anschließendes Erstellen eines Container-Images auf Grundlage des Dockerfiles. Bereitstellen des Container-Images in einem OpenShift-Cluster mit dem Befehlszeilentool `oc` von OpenShift.

- **3. Sie wurden beauftragt, auf Basis von Ruby on Rails bzw. Node.js zwei Anwendungen zum OpenShift-Cluster zu migrieren bzw. in diesem bereitzustellen. Diese Anwendungen wurden zuvor in einer Umgebung mit virtuellen Rechnern ausgeführt. Ihnen wurde der Speicherort der Git-Repositorys mitgeteilt, an dem sich der Anwendungsquellcode für beide Anwendungen befindet. Welche der folgenden Optionen wird zum Bereitstellen der Anwendungen in einem OpenShift-Cluster empfohlen? Beachten Sie dabei, dass Sie beauftragt wurden, diese Anwendungen um weitere Funktionen zu erweitern und die Entwicklung in einer OpenShift-Umgebung fortzusetzen.**
- a. Verwenden der Ruby on Rails- und Node.js-Container-Images von docker.io als Grundlage. Erstellen benutzerdefinierter Dockerfiles für die beiden Anwendungen und Erstellen von Container-Images aus ihnen. Bereitstellen der Images in einem OpenShift-Cluster mit dem StandardDeploymentprozess für Binärdateien.
 - b. Erstellen von benutzerdefinierten S2I-basierten Builder-Images, da weder für Ruby on Rails noch für Node.js in OpenShift Builder-Images verfügbar sind.
 - c. Verwenden der Ruby on Rails- und Node.js-S2I-Builder-Images aus dem Red Hat Container Catalog und Bereitstellen der Anwendungen in einem OpenShift-Cluster mit einem S2I-Build-Standardprozess.
 - d. Verwenden der einfachen Ruby- und Node.js-basierten Container-Images aus dem Red Hat Container Catalog und Erstellen von benutzerdefinierten Dockerfiles für jedes von ihnen. Erstellen und Bereitstellen der resultierenden Container-Images in einem OpenShift-Cluster.

- **4. Sie wurden beauftragt, eine in der Go-Programmiersprache geschriebene Webanwendung in einem OpenShift-Cluster bereitzustellen. Das Sicherheitsteam Ihrer Organisation möchte, dass alle Anwendungen über ein statisches Quellcode-Analysesystem und über eine Suite von automatisierten Modul- und Integrationstests ausgeführt werden, bevor sie in Produktionsumgebungen bereitgestellt werden. Das Sicherheitsteam hat zudem ein benutzerdefiniertes Dockerfile bereitgestellt. Dieses sorgt dafür, dass alle Anwendungen auf einem RHEL 7-basierten Betriebssystem basierend auf dem RHEL 7-Standard-Image aus dem Red Hat Container Catalog bereitgestellt werden. Die Umgebung enthält eine verwaltete Liste von Paketen, Benutzern und angepassten Konfigurationen für die Kernservices im Betriebssystem. Darüber hinaus hat der Anwendungsarchitekt darauf bestanden, dass es eine klare Trennung zwischen Änderungen auf Quellcodeebene und Infrastrukturänderungen (Betriebssystem, Go-Compiler und Go-Tools) gibt. Änderungen am und Patches für das Betriebssystem oder die Go-Runtime-Layer sollten automatisch eine Neuerstellung und Neubereitstellung der Anwendung auslösen. Welche der folgenden Optionen würden Sie zum Erreichen dieses Ziels verwenden?**
- a. Erstellen eines benutzerdefinierten Dockerfiles, das ein Anwendungscontainer-Image erstellt, das aus der RHEL 7-Betriebssystembasis, der Go-Laufzeit und dem Analysetool besteht. Bereitstellen des resultierenden Images in einem OpenShift-Cluster mit dem Deploymentprozess für Binärdateien.
 - b. Erstellen separater Dockerfiles für die RHEL 7-Betriebssystembasis, für die Go-Laufzeit und für das Analysetool. OpenShift kann die Dockerfiles automatisch zusammenführen, um ein einzelnes, ausführbares Anwendungscontainer-Image zu erzeugen.
 - c. Erstellen eines benutzerdefinierten S2I-Builder-Images für diese Anwendung, die das statische Analysetool, den Go-Compiler und die Laufzeit sowie das RHEL 7-Betriebssystem-Image auf Grundlage des Dockerfiles einbettet.
 - d. Erstellen separater Container-Images für die RHEL 7-Betriebssystembasis, für die Go-Laufzeit und für das Analysetool. Nachdem diese Images in einer privaten oder öffentlichen Container-Image-Registry bereitgestellt wurden, kann OpenShift die Layer aus den einzelnen Images automatisch verketten, um das endgültige ausführbare Anwendungscontainer-Image zu erstellen.
 - e. Nichts davon. Diese Anforderung kann nicht erfüllt werden und die Anwendung kann nicht in einem OpenShift-Cluster bereitgestellt werden.

Erstellen von Container-Images mit erweiterten Containerfile-Anweisungen

Ziele

Nach Abschluss dieses Abschnitts sollten Sie zu Folgendem in der Lage sein:

- Erstellen containerisierter Anwendungen mit dem Red Hat Universal Base Image
- Erstellen von Container-Images mit erweiterten Containerfile-Anweisungen

Einführung in das Red Hat Universal Base Image

Das Red Hat Universal Base Image (UBI) stellt ein hochwertiges, flexibles Basis-Container-Image für das Erstellen containerisierter Anwendungen dar. Das Ziel des Red Hat Universal Base Image besteht darin, Benutzern das Erstellen und Bereitstellen containerisierter Anwendungen mithilfe eines höchst unterstützungsfähigen, schlanken und leistungsfähigen Basis-Container-Image auf Enterprise-Niveau zu ermöglichen. Sie können Container ausführen, die mit dem Universal Base Image auf Red Hat-Plattformen sowie auf anderen Plattformen erstellt wurden.

Red Hat leitet das Red Hat Universal Base Image von Red Hat Enterprise Linux (RHEL) ab. Es unterscheidet sich von den bestehenden RHEL 7-Basis-Images, vor allem durch die Tatsache, dass es unter den Bedingungen des Red Hat Universal Base Image End User License Agreement (EULA) weiterverteilt werden kann. Das EULA ermöglicht es Partnern, Kunden und Community-Mitgliedern von Red Hat, auf gut gepflegte Unternehmenssoftware und -tools aufzubauen und durch das Hinzufügen von Inhalten Dritter Mehrwert zu schaffen.

Der Supportplan sieht vor, dass das Universal Base Image dem gleichen Lifecycle folgt und die gleichen Supportdaten wie der zugrunde liegende RHEL-Inhalt unterstützt. Wenn es auf einem abonnierten RHEL- oder OpenShift-Knoten ausgeführt wird, folgt es derselben Supportrichtlinie wie der zugrunde liegende RHEL-Inhalt. Red Hat verwaltet ein Universal Base Image für RHEL 7, das RHEL 7-Inhalten zugeordnet ist, und ein weiteres UBI für RHEL 8, das RHEL 8-Inhalten zugeordnet ist.

Red Hat empfiehlt, das Universal Base Image als Basis-Container-Image für neue Anwendungen zu verwenden. Red Hat verpflichtet sich, frühere RHEL Base Images für die Dauer des Support-Lifecycle der RHEL-Version weiterhin zu unterstützen.

Das Red Hat Universal Base Image besteht aus:

- Einem Satz von drei Basis-Images (`ubi`, `ubi-minimal` und `ubi-init`). Diese geben das wieder, was für das Erstellen von Containern mit RHEL 7-Basis-Images vorgesehen ist.
- Ein Satz mit Laufzeit-Images für Sprachen (`java`, `php`, `python`, `ruby`, `nodejs`). Diese Laufzeit-Images ermöglichen es Entwicklern, mit der Entwicklung von Anwendungen mit der Zuverlässigkeit eines von Red Hat erstelltem und unterstütztem Container-Images zu beginnen.
- Einer Reihe von zugeordneten Yum-Repositorys und Channels, die RPM-Pakete und -Updates enthalten. Damit können Sie Anwendungsabhängigkeiten hinzufügen und Container-Images bei Bedarf neu erstellen.

Typen von Universal Base Images

Das Red Hat Universal Base Image bietet drei grundlegende Basis-Images:

ubi

Ein Standard-Basis-Image, das auf RHEL-Paketen auf Enterprise-Niveau basiert. Eignet sich für die meisten Anwendungsfälle.

ubi-minimal

Ein minimales Basis-Image, das mit `microdnf` erstellt wird, einer herunterskalierten Version des `dnf`-Dienstprogramms. Dadurch wird das kleinste Container-Image angezeigt.

ubi-init

Mit diesem Image können Sie problemlos mehrere Services, z. B. Webserver, Anwendungsserver und Datenbanken, alle in einem einzigen Container ausführen. Damit können Sie das in Systemd-Moduldateien integrierte Wissen verwenden, ohne festlegen zu müssen, wie der Service gestartet werden soll.

Vorteile des Red Hat Universal Base Images

Die Verwendung des Red Hat Universal Base Images als Basis-Image für Ihre containerisierten Anwendungen hat mehrere Vorteile:

- **Minimale Größe:** Das Universal Base Image ist ein relativ minimales Basis-Container-Image (ca. 90–200 MB) mit schnellen Startzeiten.
- **Sicherheit:** Die Herkunft ist ein großes Problem bei der Verwendung von Container-Basis-Images. Sie müssen ein vertrauenswürdiges Image aus einer vertrauenswürdigen Quelle verwenden. Sprachlaufzeiten, Webserver und Kern-Libraries wie OpenSSL haben Auswirkungen auf die Sicherheit, wenn sie in die Produktion verschoben werden. Das Universal Base Image erhält zeitnahe Sicherheitsupdates von Red Hat-Sicherheitsteams.
- **Performance:** Die Basis-Images werden von einem internen Performance Engineering Team bei Red Hat getestet, optimiert und zertifiziert. Hierbei handelt es sich um bewährte Container-Images, die in einigen der rechenintensivsten, E/A-intensivsten und fehlerempfindlichsten Workloads der Welt ausgiebig verwendet werden.
- **ISV, Anbieterzertifizierung und Partnersupport:** Das Universal Base Image erbt das breite Ökosystem von RHEL-Partnern, ISVs und Drittanbietern, die Tausende von Anwendungen unterstützen. Das Universal Base Image erleichtert diesen Partnern das Erstellen, Bereitstellen und Zertifizieren ihrer Anwendungen und ermöglicht es ihnen, die resultierende containerisierte Anwendung sowohl auf Red Hat-Plattformen wie RHEL und OpenShift als auch auf anderen Plattformen bereitzustellen.
- **Einmal erstellen, auf vielen verschiedenen Hosts bereitstellen:** Das Red Hat Universal Base Image kann überall erstellt und bereitgestellt werden: auf OpenShift/RHEL oder einem anderen Containerhost (Fedora, Debian, Ubuntu und mehr).

Erweiterte Containerfile-Anweisungen

Ein Containerfile automatisiert das Erstellen von Container-Images. Ein Containerfile ist eine Textdatei, die eine Reihe von Anweisungen zum Erstellen des Container-Images enthält. Die Anweisungen werden nacheinander, in sequenzieller Reihenfolge ausgeführt. In diesem Abschnitt werden einige der grundlegenden Containerfile-Anweisungen beschrieben und anschließend einige erweiterte Anweisungen erläutert, einschließlich praktischer Möglichkeiten, sie beim Erstellen von Container-Images für die Bereitstellung in OpenShift zu verwenden.

Die RUN-Anweisung

Die Anweisung RUN führt Befehle auf einem neuen, dem aktuellen Image übergeordneten Layer aus und übergibt anschließend die Ergebnisse. Der Container-Build-Prozess verwendet dann

Kapitel 2 | Entwerfen containerisierter Anwendungen für OpenShift

das übergebene Ergebnis im nächsten Schritt in dem Containerfile. Der Container-Build-Prozess verwendet /bin/sh zum Ausführen von Befehlen.

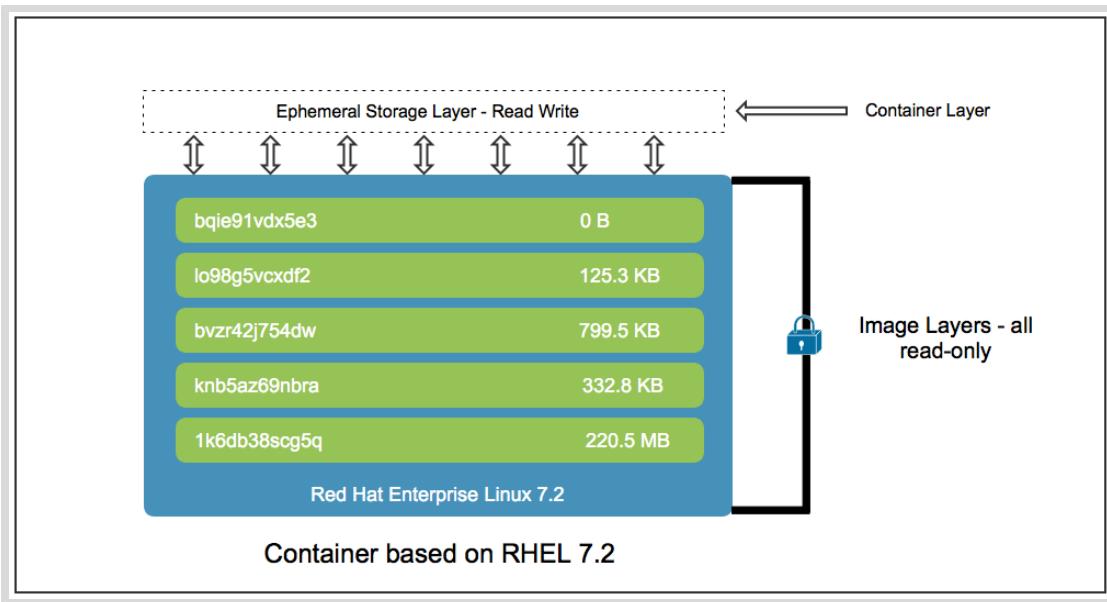


Abbildung 2.1: Layer in einem Container-Image

Jede Anweisung in einem Containerfile führt zu einem neuen Layer für das endgültige Container-Image. Zu viele Anweisungen in einem Containerfile bedingen daher ein Übermaß an Layern, wodurch die Images unnötig umfangreich werden. Betrachten Sie beispielsweise die folgende RUN-Anweisung in einem Containerfile:

```
RUN yum --disablerepo=* --enablerepo="rhel-7-server-rpms"  
RUN yum update  
RUN yum install -y httpd  
RUN yum clean all -y
```

Die vorherige Vorgehensweise eignet sich nicht für die Erstellung von Container-Images, da vier Layer für denselben Zweck erstellt werden. Beim Erstellen von Containerfiles empfiehlt Red Hat die Anzahl der Layer zu minimieren. Sie können dieselben Ziele erreichen, indem Sie das Befehlstrennzeichen && verwenden, um mehrere Befehle mit einer einzigen RUN-Anweisung auszuführen:

```
RUN yum --disablerepo=* --enablerepo="rhel-7-server-rpms" && \  
    yum update && \  
    yum install -y httpd && \  
    yum clean all -y
```

In diesem aktualisierten Beispiel wird nur ein Layer erstellt und die Lesbarkeit nicht kompromittiert.



Anmerkung

In Podman erstellen nur die Anweisungen RUN, COPY und ADD Layer . Andere Anweisungen erstellen temporäre Zwischen-Images und erhöhen nicht direkt die Größe des Images.

Die LABEL-Anweisung

Die Anweisung `LABEL` definiert Image-Metadaten. Bei Bezeichnungen (Labels) handelt es sich um an ein Image angefügte Schlüssel-Wert-Paare. Die Anweisung `LABEL` fügt dem Image in der Regel beschreibende Metadaten hinzu. Dazu zählen beispielsweise Versionen, eine kurze Beschreibung und andere Details, die Benutzern des Images Informationen bereitstellen.

Wenn Sie Images für OpenShift erstellen, geben Sie vor dem Bezeichnungsnamen das Präfix `io.openshift` an, um zwischen OpenShift- und Kubernetes-bezogenen Metadaten zu unterscheiden. Das OpenShift-Tooling kann bestimmte Bezeichnungen analysieren und bestimmte Aktionen anhand des Vorhandenseins dieser Bezeichnungen durchführen. In der folgenden Tabelle werden einige der am häufigsten verwendeten Tags aufgeführt:

Von OpenShift unterstützte Bezeichnungen

Bezeichnungsname	Beschreibung
<code>io.openshift.tags</code>	Diese Bezeichnung enthält eine Liste von durch Komma getrennten Tags. Tags kategorisieren Container-Images in umfassende Funktionsbereiche. Anhand von Tags können Benutzeroberflächen- und Generierungstools, während des Anwendungserstellungsprozesses relevante Container-Images vorschlagen.
<code>io.k8s.description</code>	Diese Bezeichnung liefert Nutzern von Container-Images weitere Details zum Service oder zur Funktion, der bzw. die von diesem Image bereitgestellt wird.
<code>io.openshift.expose-services</code>	Diese Bezeichnung enthält eine Liste der Serviceports, die mit den EXPOSE-Anweisungen im Containerfile übereinstimmen, sowie weitere beschreibende Informationen darüber, was der jeweilige Service Nutzern bereitstellt. Das Format lautet <code>PORT[/PROTO]:NAME</code> , wobei der Teil <code>[/PROTO]</code> optional ist und standardmäßig <code>tcp</code> lautet, wenn keine Angabe erfolgt ist.

Eine vollständige Liste von allen OpenShift-spezifischen Bezeichnungen, ihre Beschreibungen und eine Beispieldatenfindung finden Sie in den Referenzen am Ende dieses Abschnitts.

Die WORKDIR-Anweisung

Die Anweisung `WORKDIR` legt das Arbeitsverzeichnis für die Anweisungen `RUN`, `CMD`, `ENTRYPOINT`, `COPY` oder `ADD` in einem Containerfile fest.

Red Hat empfiehlt, absolute Pfade in `WORKDIR`-Anweisungen zu verwenden. Verwenden Sie `WORKDIR` anstelle von mehreren `RUN`-Anweisungen, wenn Sie Verzeichnisse ändern und dann einige Befehle ausführen. Dieser Ansatz gewährleistet langfristig eine bessere Verwaltbarkeit. Zudem lassen sich dadurch Fehler leichter beheben.

Die ENV-Anweisung

Die Anweisung `ENV` definiert Umgebungsvariablen, die dem Container zur Verfügung stehen. Sie können in einem Containerfile mehrere `ENV`-Anweisungen deklarieren. Sie können mithilfe des Befehls `env` im Container alle Umgebungsvariablen anzeigen.

Es empfiehlt sich, ENV-Anweisungen zum Definieren von Datei- und Ordnerpfaden zu verwenden, anstatt sie in den Containerfile-Anweisungen hart zu codieren. Es ist nützlich, Informationen wie Softwareversionsnummern zu speichern und auch Verzeichnisse an die Umgebungsvariable PATH anzuhängen.

Die USER-Anweisung

Red Hat empfiehlt, dass Sie das Image aus Sicherheitsgründen nicht als Root-Benutzer ausführen. Vermeiden Sie, die USER-Anweisung in einem Containerfile zu oft zu verwenden. Auf diese Weise reduzieren Sie die Anzahl der Layer . Die Sicherheitsbelange in Bezug auf das Ausführen von Containern als Nicht-Root-Benutzer finden Sie weiter unten in diesem Abschnitt.



Warnung

Von OpenShift wird die vom Container-Image festgelegte USER-Anweisung standardmäßig nicht berücksichtigt. Aus Sicherheitsgründen verwendet OpenShift zum Ausführen von Containern eine zufällige Benutzer-ID (userid), die sich von der Root-Benutzer-ID (root userid) (0) unterscheidet.

Die VOLUME-Anweisung

Die Anweisung VOLUME erstellt einen Mount-Punkt im Container und gibt Image-Nutzern an, dass extern gemountete Volumes vom Host-Rechner oder anderen Containern an diesen Mount-Punkt gebunden werden können.

Red Hat empfiehlt, VOLUME-Anweisungen für persistente Daten zu verwenden. OpenShift kann den netzgebundenen Speicher am Knoten mounten, auf dem der Container ausgeführt wird. Wenn der Container auf einen neuen Knoten verschoben wird, dann wird der Storage erneut an den Knoten angefügt. Durch Nutzung des Volumes für alle persistenten Storage-Anforderungen behalten Sie den Inhalt bei, selbst wenn der Container neu gestartet oder verschoben wird.

Darüber hinaus können durch das Definieren von Volumes in Ihrem Containerfile Image-Nutzer ohne Weiteres verstehen, welche Volumes sie definieren können, wenn Ihr Image ausgeführt wird.

Erstellen von Images mit der ONBUILD-Anweisung

Die Anweisung ONBUILD registriert *Trigger* im Container-Image. Ein Containerfile verwendet ONBUILD, um Anweisungen zu deklarieren, die nur beim Erstellen eines untergeordneten Images ausgeführt werden.

Mit der Anweisung ONBUILD kann eine einfache Anpassung eines Container-Images für gängige Anwendungsfälle unterstützt werden. Dazu zählen beispielsweise das Vorabladen von Daten oder das Bereitstellen einer benutzerdefinierten Konfiguration für eine Anwendung. Das übergeordnete Image stellt für alle untergeordneten Downstream-Images gängige Befehle bereit. Das untergeordnete Image stellt nur die Daten und Konfigurationsdateien bereit. Das Containerfile für das untergeordnete Image kann so einfach sein, dass es nur die FROM-Anweisung enthält, die auf das übergeordnete Image verweist.



Anmerkung

Die Anweisung ONBUILD ist nicht in der OCI-Spezifikation enthalten und wird daher standardmäßig nicht unterstützt, wenn Sie Container mit Podman oder Buildah erstellen. Verwenden Sie die Option `--format docker`, um die Unterstützung für die Anweisung ONBUILD zu aktivieren.

Kapitel 2 | Entwerfen containerisierter Anwendungen für OpenShift

Angenommen, Sie erstellen ein übergeordnetes Node.js-Image, das alle Entwickler in Ihrer Organisation als Grundlage verwenden sollen, wenn sie Anwendungen mit den folgenden Anforderungen erstellen:

- Erzwingen bestimmter Standards, beispielsweise das Kopieren von JavaScript-Quellen in den Anwendungsordner, damit sie von der Node.js-Engine interpretiert werden.
- Ausführen des Befehls `npm install`, der in der Datei `package.json` beschriebene Abhängigkeiten abruft.

Sie können diese Anforderungen nicht als Anweisungen im übergeordneten Containerfile einbetten, da Sie den Anwendungskode nicht besitzen und die jeweilige Anwendung möglicherweise unterschiedliche Abhängigkeiten aufweist, die in der Datei `package.json` aufgelistet sind.

Deklarieren Sie `ONBUILD`-Anweisungen im übergeordneten Containerfile. Das übergeordnete Containerfile ist im Folgenden dargestellt:

```
FROM registry.access.redhat.com/rhscl/nodejs-6-rhel7
EXPOSE 3000
# Make all Node.js apps use /opt/app-root as the main folder (APP_ROOT).
RUN mkdir -p /opt/app-root/
WORKDIR /opt/app-root

# Copy the package.json to APP_ROOT
ONBUILD COPY package.json /opt/app-root

# Install the dependencies
ONBUILD RUN npm install

# Copy the app source code to APP_ROOT
ONBUILD COPY src /opt/app-root

# Start node server on port 3000
CMD [ "npm", "start" ]
```

Angenommen, Sie haben das übergeordnete Container-Image erstellt und `mynodejs-base` genannt. In diesem Fall würde ein untergeordnetes Containerfile für eine Anwendung, die das übergeordnete Image verwendet, wie folgt aussehen:

```
FROM mynodejs-base
RUN echo "Started Node.js server..."
```

Beim Starten des Build-Prozesses für das untergeordnete Image wird die Ausführung der drei im übergeordneten Image definierten `ONBUILD`-Anweisungen ausgelöst, bevor die `RUN`-Anweisung im untergeordneten Containerfile aufgerufen wird.

Überlegungen zu OpenShift für die USER-Anweisung

OpenShift führt Container standardmäßig mit einer beliebig zugewiesenen „userid“ aus. Durch diesen Ansatz wird das Risiko reduziert, dass im Container ausgeführte Prozesse aufgrund von Sicherheitslücken in der Container-Engine erhöhte Berechtigungen auf dem Host-Rechner erhalten.

Anpassen von Containerfiles für OpenShift

Beachten Sie beim Schreiben oder Ändern eines Containerfiles, das ein Image für die Ausführung in einem OpenShift-Cluster erstellt, Folgendes:

- Verzeichnisse und Dateien, die von Prozessen im Container ausgelesen oder in Prozesse im Container geschrieben werden, sollten Eigentum der Gruppe `root` sein und über Lese- und Schreibberechtigungen für die Gruppe verfügen.
- Ausführbare Dateien sollten Ausführungsberechtigungen für die Gruppe aufweisen.
- Die im Container ausgeführten Prozesse dürfen privilegierte Ports (d. h. Ports unter 1024) nicht überwachen, da sie nicht als privilegierte Benutzer ausgeführt werden.

Durch das Hinzufügen der folgenden RUN-Anweisung zu Ihrem Containerfile werden die Verzeichnis- und Dateiberechtigungen so festgelegt, dass Benutzer in der Gruppe `root` auf diese im Container zugreifen können:

```
RUN chgrp -R 0 directory && \
    chmod -R g=u directory
```

Das Benutzerkonto, das den Container ausführt, ist immer Mitglied der Gruppe `root`. Folglich kann der Container dieses Verzeichnis lesen und in das Verzeichnis schreiben. Die Gruppe `root` weist keine speziellen Berechtigungen auf (im Gegensatz zum Benutzer `root`), was die Sicherheitsrisiken bei dieser Konfiguration minimiert.

Durch das Argument `g=u` aus dem Befehl `chmod` werden die Berechtigungen vom Typ „group“ identisch mit den Berechtigungen vom Typ „owner user“. Diese sind standardmäßig Lese- und Schreibberechtigungen. Mit dem Argument `g+rwx` erzielen Sie dieselben Ergebnisse.

Ausführen von Containern als „root“ mit Sicherheitskontextbeschränkungen (Security Context Constraints, SCC)

In bestimmten Fällen haben Sie möglicherweise keinen Zugriff auf Containerfiles für bestimmte Images. Sie müssen das Image möglicherweise als Benutzer `root` ausführen. In diesem Szenario müssen Sie OpenShift so konfigurieren, dass Container als `root` ausgeführt werden können.

OpenShift bietet *Sicherheitskontextbeschränkungen (Security Context Constraints, SCCs)*, die steuern, welche Aktionen ein Pod ausführen und auf welche Ressourcen er zugreifen kann. OpenShift enthält standardmäßig eine Reihe integrierter SCCs. Alle von OpenShift erstellten Container verwenden standardmäßig die SCC `restricted`. Diese ignoriert die vom Container-Image festgelegte „userid“ und weist Containern eine zufällige „userid“ zu.

Damit Container eine feste „userid“ wie 0 (der Benutzer `root`) verwenden können, müssen Sie die SCC `anyuid` verwenden. Dazu müssen Sie zunächst ein Servicekonto erstellen. Ein Servicekonto ist die OpenShift-Identität für einen Pod. Alle Pods aus einem Projekt werden unter einem Standardservicekonto ausgeführt, sofern der Pod oder dessen Deploymentkonfiguration nicht anderweitig konfiguriert ist.

Wenn Sie für eine Anwendung eine von der SCC „restricted“ nicht gewährte Fähigkeit erforderlich ist, sollten Sie ein neues spezifisches Servicekonto erstellen, der entsprechenden SCC hinzufügen und die Deploymentkonfiguration ändern, die die Anwendungs-Pods erstellt, um das neue Servicekonto zu verwenden.

In den folgenden Schritten wird detailliert beschrieben, wie festgelegt werden kann, dass Container als Benutzer `root` in einem OpenShift-Projekt ausgeführt werden:

- Erstellen Sie ein neues Servicekonto:

```
[user@host ~]$ oc create serviceaccount myserviceaccount
```

- Ändern Sie die Deploymentkonfiguration für die Anwendung, um das neue Servicekonto zu verwenden: Führen Sie dazu den Befehl `oc patch` aus:

```
[user@host ~]$ oc patch dc/demo-app --patch \  
'{"spec": {"template": {"spec": {"serviceAccountName": "myserviceaccount"}}}}'
```



Anmerkung

Details zur Verwendung des Befehls `oc patch` finden Sie unter OpenShift admins guide to `oc patch` [<https://access.redhat.com/articles/3319751>]. Führen Sie den Befehl `oc patch -h` aus, um Informationen zur Verwendung anzuzeigen.

- Fügen Sie der SCC `anyuid` das Servicekonto `myserviceaccount` hinzu, um sie mit einer festen „userid“ im Container auszuführen:

```
[user@host ~]$ oc adm policy add-scc-to-user anyuid -z myserviceaccount
```



Literaturhinweise

Best Practices für das Schreiben von Dockerfiles

https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices

Weitere Informationen zum Erstellen von Images finden Sie im Kapitel *Creating Images* des Handbuchs *Images* der Produktdokumentation für OpenShift Container Platform unter

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/images/creating-images

► Angeleitete Übung

Erstellen von Container-Images mit erweiterten Containerfile-Anweisungen

In dieser Übung verwenden Sie Red Hat OpenShift zum Erstellen und Bereitstellen eines Apache HTTP Server-Containers aus einem Containerfile.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Erstellen eines Apache HTTP Server-Container-Images mit einem Containerfile und Bereitstellen auf einem OpenShift-Cluster
- Erstellen eines untergeordneten Container-Images durch das Erweitern des übergeordneten Apache HTTP Server-Images
- Ändern des Containerfiles für das untergeordnete Container-Image, sodass es in einem OpenShift-Cluster mit einer zufälligen Benutzer-ID ausgeführt wird

Bevor Sie Beginnen

Zum Durchführen dieser Aufgabe benötigen Sie Zugriff:

- Auf einen aktiven OpenShift-Cluster
- Auf das übergeordnete Image für Apache HTTP Server (quay.io/redhattraining/httpd-parent).
- Auf das Containerfile für das untergeordnete Container-Image im Git-Repository (`container-build`)

Führen Sie den folgenden Befehl auf der `workstation`-VM aus, um die Voraussetzungen zu überprüfen und die Lösungsdateien herunterzuladen:

```
[student@workstation ~]$ lab container-build start
```

Anweisungen

► 1. Überprüfen Sie das übergeordnete Containerfile für Apache HTTP Server.

Ein vorab erstelltes übergeordnetes Container-Image für Apache HTTP Server befindet sich in der öffentlichen Quay.io-Registry unter `quay.io/redhattraining/httpd-parent`. Überprüfen Sie kurz das Containerfile für dieses übergeordnete Image, das sich unter `~/D0288/labs/container-build/httpd-parent/Containerfile` befindet:

```
FROM registry.access.redhat.com/ubi8/ubi:8.0 ①
```

```
MAINTAINER Red Hat Training <training@redhat.com>
```

```
# DocumentRoot for Apache
ENV DOCROOT=/var/www/html ②

RUN  yum install -y --no-docs --disableplugin=subscription-manager httpd && \
      yum clean all --disableplugin=subscription-manager -y && \
      echo "Hello from the httpd-parent container!" > ${DOCROOT}/index.html ③

# Allows child images to inject their own content into DocumentRoot
ONBUILD COPY src/ ${DOCROOT}/ ④

EXPOSE 80

# This stuff is needed to ensure a clean start
RUN rm -rf /run/httpd && mkdir /run/httpd

# Run as the root user
USER root ⑤

# Launch httpd
CMD /usr/sbin/httpd -DFOREGROUND
```

- ① Das Basis-Image ist das Universal Base Image (UBI) für Red Hat Enterprise Linux 8.0 aus dem Red Hat Container Catalog.
- ② Umgebungsvariablen für dieses Container-Image.
- ③ Die Anweisung RUN enthält verschiedene Befehle, die Apache HTTP Server installieren und eine standardmäßige Startseite für den Webserver erstellen.
- ④ Durch die ONBUILD-Anweisung können untergeordnete Images eigene benutzerdefinierte Webserverinhalte bereitstellen, wenn ein Image erstellt wird, das von diesem übergeordneten Image erweitert wird.
- ⑤ Die USER-Anweisung führt den Apache HTTP Server-Prozess als Benutzer root aus.



Anmerkung

Beachten Sie, dass die RUN-Zeilen möglichst verschiedene Befehle in einer einzelnen Anweisung kombinieren, um die Anzahl der Layer im Image zu reduzieren. Dies führt zu kleineren Images, die schneller bereitgestellt werden können.

► 2. Überprüfen Sie das untergeordnete Containerfile für Apache HTTP Server.

Verwenden Sie das übergeordnete Apache HTTP Server-Container-Image (`redhattraining/httpd-parent`) als eine Grundlage zum Erweitern und Anpassen des Images an Ihre Anwendung. Das Containerfile für das untergeordnete Container-Image wird auf dem Git-Repository-Server des Kursraums gespeichert. Führen Sie zum Anzeigen des Containerfiles die folgenden Schritte aus:

2.1. Laden Sie die Konfiguration Ihrer Kursumgebung.

Führen Sie den folgenden Befehl aus, um die in der ersten angeleiteten Übung erstellten Umgebungsvariablen zu laden:

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

- 2.2. Wechseln Sie zu Ihrem lokalen Klon des D0288-apps-Git-Repository und checken Sie den main-Branch des Kurs-Repository aus, um sicherzustellen, dass Sie diese Übung von einem als fehlerfrei bekannten Zustand aus starten:

```
[student@workstation ~]$ cd D0288-apps
[student@workstation D0288-apps]$ git checkout main
...output omitted...
```

- 2.3. Erstellen Sie einen neuen Branch, in dem Sie alle Änderungen speichern können, die Sie während dieser Übung vornehmen:

```
[student@workstation D0288-apps]$ git checkout -b container-build
Switched to a new branch 'container-build'
[student@workstation D0288-apps]$ git push -u origin container-build
...output omitted...
* [new branch]      container-build -> container-build
Branch container-build set up to track remote branch container-build from origin.
```

- 2.4. Überprüfen Sie die Datei ~/D0288-apps/container-build/Containerfile. Das Containerfile enthält eine einzige Anweisung, FROM, die das Image redhattraining/httpd-parent verwendet:

```
FROM quay.io/redhattraining/http-parent
```

- 2.5. Der untergeordnete Container stellt eine eigene Datei index.html im Ordner ~/D0288-apps/container-build/src bereit, wodurch die Datei index.html des übergeordneten Containers überschrieben wird. Der Inhalt der Datei index.html des untergeordneten Container-Images ist unten dargestellt:

```
<!DOCTYPE html>
<html>
<body>
  Hello from the Apache child container!
</body>
</html>
```

- 3. Erstellen Sie mithilfe des untergeordneten Containerfiles für Apache HTTP Server einen Container und stellen Sie ihn in einem OpenShift-Cluster bereit.

- 3.1. Melden Sie sich bei OpenShift mit Ihrem Entwicklerbenutzernamen an:

```
[student@workstation D0288-apps]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

- 3.2. Erstellen Sie ein neues Projekt für die Anwendung. Stellen Sie dem Projektnamen Ihren Entwicklerbenutzernamen voran.

```
[student@workstation D0288-apps]$ oc new-project \
${RHT_OCP4_DEV_USER}-container-build
Now using project "youruser-container-build" on server "https://
api.cluster.domain.example.com:6443".
```

3.3. Erstellen Sie das untergeordnete Image für Apache HTTP Server:

```
[student@workstation D0288-apps]$ podman build --layers=false \
-t do288-apache ./container-build
STEP 1: FROM quay.io/redhattraining/httpd-parent ①
...output omitted...
STEP 2: COPY src/ ${DOCROOT}/ ②
STEP 3: COMMIT do288-apache
...output omitted...
Storing signatures
...output omitted...
```

- ① Das Containerfile wird automatisch identifiziert, und podman pull ist das übergeordnete Image.
- ② Die Anweisung ONBUILD im übergeordneten Containerfile löst das Kopieren der Datei `index.html` des untergeordneten Elements aus, wodurch die übergeordnete Indexdatei überschrieben wird.



Wichtig

Der Start des Build-Prozesses kann einige Zeit in Anspruch nehmen. Wenn die folgende Ausgabe angezeigt wird, führen Sie den Befehl erneut aus.

```
Error from server (BadRequest): unable to wait for build hola-1 to run: timed
out waiting for the condition
```

3.4. Zeigen Sie die Images an:

```
[student@workstation D0288-apps]$ podman images
localhost/do288-apache           latest fc114a884288 9 minutes ago 236 MB
quay.io/redhattraining/httpd-parent latest 4346d3cace25 2 years ago 236 MB
```

3.5. Versehen Sie das Image mit einem Tag, und übertragen Sie es an Quay.io.

```
[student@workstation D0288-apps]$ podman tag do288-apache \
quay.io/${RHT_OCP4_QUAY_USER}/do288-apache
[student@workstation D0288-apps]$ podman login quay.io -u ${RHT_OCP4_QUAY_USER}
Password:
Login Succeeded!
[student@workstation D0288-apps]$ podman push \
quay.io/${RHT_OCP4_QUAY_USER}/do288-apache
...output omitted...
Storing signatures
[student@workstation D0288-apps]$
```

- 3.6. Melden Sie sich Quay.io [https://quay.io] bei an, und veröffentlichen Sie das neue Image.

- 3.7. Stellen Sie das untergeordnete Apache HTTP Server-Image bereit:

```
[student@workstation D0288-apps]$ oc new-app --name hola \
quay.io/${RHT_OCP4_QUAY_USER}/do288-apache
--> Found Container image 1d6f8d3 (11 minutes old) from quay.io for "quay.io/
youruser/do288-apache"

Red{nbsp}Hat Universal Base Image 8
-----
The Universal Base Image is designed and engineered to be the base layer
for all of your containerized applications, middleware and utilities. This base
image is freely redistributable, but Red{nbsp}Hat only supports Red{nbsp}Hat
technologies through subscriptions for Red{nbsp}Hat products. This image is
maintained by Red{nbsp}Hat and updated regularly.

Tags: base rhel8

...output omitted...
--> Success
Application is not exposed. You can expose services to the outside world by
executing one or more of the commands below:
'oc expose svc/hola'
Run 'oc status' to view your app.
```

- 4. Vergewissern Sie sich, dass der Anwendungs-Pod nicht gestartet wurde. Der Pod weist den Zustand **Error** auf. Wenn Sie jedoch zu lange warten, wechselt der Pod in den Zustand **CrashLoopBackOff**.

```
[student@workstation D0288-apps]$ oc get pods
NAME          READY   STATUS      RESTARTS   AGE
hola-13p75f5   0/1     CrashLoopBackOff   0          12s
```

- 5. Überprüfen Sie die Logs des Containers, um zu ermitteln, weshalb der Pod nicht gestartet werden konnte:

```
[student@workstation D0288-apps]$ oc logs hola-13p75f5
AH00558: httpd: Could not reliably determine the server's fully qualified domain
           name...
(13)Permission denied: AH00072: make_sock: could not bind to address [::]:80 ①
(13)Permission denied: AH00072: make_sock: could not bind to address 0.0.0.0:80 ②
no listening sockets available, shutting down
AH00015: Unable to open logs ③
```

- ① Da OpenShift Container mit einer zufälligen „userid“ ausführt, sind die Ports unter 1024 privilegiert und können nur als **root** ausgeführt werden.
- ② Die von OpenShift zum Ausführen des Containers verwendete zufällige „userid“ besitzt keine Berechtigungen zum Lesen und Schreiben von Logdateien in **/var/log/httpd** (der standardmäßige Logdateispeicherort für Apache HTTP Server in RHEL 7).



Warnung

Der fehlerhafte Anwendungs-Pod wird nach kurzer Zeit gelöscht. Achten Sie darauf, die Anwendungs-Pod-Logdateien zu überprüfen, bevor der Pod gelöscht wird.

- 6. Löschen Sie alle Ressourcen aus dem OpenShift-Projekt. Im nächsten Schritt wird das Containerfile entsprechend den Red Hat-Empfehlungen für OpenShift geändert.

Löschen Sie vor dem Aktualisieren des untergeordneten Containerfiles für Apache HTTP Server alle Ressourcen im Projekt, die bisher erstellt wurden:

```
[student@workstation D0288-apps]$ oc delete all -l app=hola
service "hola" deleted
deployment.apps "hola" deleted
build.build.openshift.io "hola-1" deleted
```

- 7. Ändern Sie das Containerfile für den untergeordneten Container, damit es in einem OpenShift-Cluster ausgeführt werden kann, indem Sie den Apache HTTP Server-Prozess so aktualisieren, dass er als zufälliger unprivilegierter Benutzer ausgeführt wird.

- 7.1. Bearbeiten Sie die Datei ~/D0288-apps/container-build/Containerfile, und führen Sie die folgenden Schritte aus. Sie können diese Anweisungen auch aus der bereitgestellten Datei ~/D0288/solutions/container-build/Containerfile kopieren.
- 7.2. Überschreiben Sie die EXPOSE-Anweisung aus dem übergeordneten Image und ändern Sie den Port in 8080.

```
EXPOSE 8080
```

- 7.3. Beziehen Sie die Bezeichnung io.openshift.expose-service ein, um den geänderten Port anzugeben, auf dem der Webserver ausgeführt wird:

```
LABEL io.openshift.expose-services="8080:http"
```

Aktualisieren Sie die Liste der Bezeichnungen, um die Bezeichnungen io.k8s.description, io.k8s.display-name und io.openshift.tags einzubeziehen, über die OpenShift hilfreiche Metadaten zum Container-Image bereitstellt:

```
LABEL io.k8s.description="A basic Apache HTTP Server child image, uses ONBUILD" \
io.k8s.display-name="Apache HTTP Server" \
io.openshift.expose-services="8080:http" \
io.openshift.tags="apache, httpd"
```

- 7.4. Sie müssen den Webserver auf einem unprivilegierten Port (d. h. größer als 1024) ausführen. Ändern Sie mit einer RUN-Anweisung die Portnummer in der Konfigurationsdatei von Apache HTTP Server vom Standardport 80 in 8080:

```
RUN sed -i "s/Listen 80/Listen 8080/g" /etc/httpd/conf/httpd.conf  
RUN sed -i "s/#ServerName www.example.com:80/ServerName 0.0.0.0:8080/g" /etc/  
httpd/conf/httpd.conf
```

- 7.5. Ändern Sie die Gruppen-ID und Berechtigungen der Ordner, in denen der Webserverprozess Dateien liest bzw. in die er schreibt:

```
RUN chgrp -R 0 /var/log/httpd /var/run/httpd && \  
chmod -R g=u /var/log/httpd /var/run/httpd
```

- 7.6. Fügen Sie eine USER-Anweisung für einen unprivilegierten Benutzer hinzu. Per Red Hat-Konvention wird die „userid“ 1001 verwendet:

```
USER 1001
```

- 7.7. Speichern Sie das Containerfile und übergeben Sie die Änderungen aus dem Ordner ~/D0288-apps/container-build an das Git-Repository:

```
[student@workstation D0288-apps]$ cd container-build  
[student@workstation container-build]$ git commit -a -m \  
"Changed the Containerfile to enable running as a random uid on OpenShift"  
...output omitted...  
[student@workstation container-build]$ git push  
...output omitted...  
[student@workstation container-build]$ cd ..
```

- 8. Erstellen Sie das untergeordnete Container-Image für Apache HTTP Server neu und stellen Sie es erneut bereit:

- 8.1. Alte Images entfernen

```
[student@workstation ~]$ podman rmi -a --force  
...output omitted...
```

- 8.2. Erstellen Sie die Anwendung mit dem neuen Containerfile erneut:

```
[student@workstation ~]$ podman build --layers=false \  
-t do288-apache ./container-build  
STEP 1: FROM quay.io/redhattraining/httpd-parent  
...output omitted...  
STEP 2: COPY src/ ${DOCROOT}/  
STEP 3: EXPOSE 8080  
STEP 4: LABEL io.k8s.description="A basic Apache HTTP Server child image,  
uses ONBUILD" io.k8s.display-name="Apache HTTP Server"  
io.openshift.expose-services="8080:http" io.openshift.tags="apache, httpd"  
STEP 5: RUN sed -i "s/Listen 80/Listen 8080/g" /etc/httpd/conf/httpd.conf  
STEP 6: RUN chgrp -R 0 /var/log/httpd /var/run/httpd && chmod -R g=u /var/log/  
httpd /var/run/httpd
```

```
STEP 7: USER 1001  
STEP 8: COMMIT do288-apache  
...output omitted...
```

- 8.3. Kennzeichnen Sie das neue Image und ersetzen Sie das Image in Quay.io

```
[student@workstation D0288-apps]$ podman tag do288-apache \  
quay.io/${RHT_OCP4_QUAY_USER}/do288-apache  
[student@workstation D0288-apps]$ podman push \  
quay.io/${RHT_OCP4_QUAY_USER}/do288-apache  
...output omitted...  
Storing signatures  
[student@workstation D0288-apps]$
```

- 8.4. Stellen Sie das untergeordnete Container-Image für Apache HTTP Server erneut bereit:

```
[student@workstation ~]$ oc new-app --name hola \  
quay.io/${RHT_OCP4_QUAY_USER}/do288-apache  
--> Found container image fe746d5 (7 minutes old) from quay.io for "quay.io/  
youruser/do288-apache"  
...output omitted...  
--> Success  
Application is not exposed. You can expose services to the outside world by  
executing one or more of the commands below:  
'oc expose service/hola'  
Run 'oc status' to view your app.
```

- 8.5. Warten Sie, bis der Pod bereit ist und ausgeführt wird. Zeigen Sie den Status des Anwendungs-Pods an:

```
[student@workstation ~]$ oc get pods  
NAME          READY   STATUS    RESTARTS   AGE  
hola-1775gkw  1/1     Running   0          5s
```

Der Anwendungs-Pod wird nun erfolgreich gestartet und befindet sich im Zustand Running.

- 9. Erstellen Sie eine OpenShift-Route, um die Anwendung für den externen Zugriff bereitzustellen:

```
[student@workstation ~]$ oc expose --port='8080' svc/hola  
route.route.openshift.io/hola exposed
```

- 10. Rufen Sie die Routen-URL mit dem Befehl `oc get route` ab:

```
[student@workstation ~]$ oc get route  
NAME      HOST/PORT                                     PATH      SERVICES  
PORT      TERMINATION      WILDCARD  
hola     hola-youruser-container-build.cluster.domain.example.com      hola  
8080-tcp           None
```

Kapitel 2 | Entwerfen containerisierter Anwendungen für OpenShift

- 11. Testen Sie die Anwendung mit der Routen-URL, die Sie im vorherigen Schritt abgerufen haben:

```
[student@workstation ~]$ curl \
http://hola-${RHT_OCP4_DEV_USER}-container-build.${RHT_OCP4_WILDCARD_DOMAIN}
...output omitted...
Hello from the Apache child container!
...output omitted...
```

- 12. Führen Sie eine Bereinigung durch. Löschen Sie das Projekt.

```
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-container-build
```

Beenden

Führen Sie auf der `workstation` den Befehl `lab container-build finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken.

```
[student@workstation ~]$ lab container-build finish
```

Hiermit ist die angeleitete Übung beendet.

Einfügen von Konfigurationsdaten in eine Anwendung

Ziele

Nach Abschluss dieses Abschnitts sollten Sie in der Lage sein, eine Methode zum Einfügen von Konfigurationsdaten in eine Anwendung auszuwählen und die dazu erforderlichen Ressourcen zu erstellen.

Externalisieren der Anwendungskonfiguration in OpenShift

Entwickler konfigurieren ihre Anwendungen in der Regel durch eine Kombination aus Umgebungsvariablen, Befehlszeilenargumenten und Konfigurationsdateien. Beim Bereitstellen von Anwendungen in OpenShift stellt die Konfigurationsverwaltung aufgrund der unveränderbaren Art der Container eine Herausforderung dar. Im Gegensatz zu herkömmlichen, nicht containerisierten Bereitstellungen wird es nicht empfohlen, bei der Ausführung von containerisierten Anwendungen die Anwendung mit der Konfiguration zu koppeln.

Der empfohlene Ansatz für containerisierte Anwendungen besteht darin, die statischen Anwendungsbinärdateien von den dynamischen Konfigurationsdaten zu entkoppeln und die Konfiguration zu externalisieren. Durch diese Trennung wird die Portabilität der Anwendungen über viele Umgebungen hinweg gewährleistet.

Angenommen, Sie möchten eine in einem OpenShift-Cluster bereitgestellte Anwendung aus einer Entwicklungsumgebung in eine Produktionsumgebung mit Zwischenphasen wie Tests und Benutzerakzeptanz heraufstufen. Sie sollten dasselbe Anwendungscontainer-Image in allen Phasen verwenden. Zudem sollten die Konfigurationsdetails für die jeweilige Umgebung außerhalb des Container-Images spezifisch sein.

Verwenden von Secret- und ConfigMap-Ressourcen

OpenShift stellt Secret- und ConfigMap-Ressourcentypen zum Externalisieren und Verwalten von Konfigurationen für Anwendungen bereit.

Secret-Ressourcen werden zum Speichern sensibler Informationen wie Passwörter, Schlüssel und Token verwendet. Als Entwickler ist es wichtig, Secrets zu erstellen, um zu vermeiden, dass Anmelddaten und andere vertrauliche Informationen in Ihrer Anwendung kompromittiert werden. Es gibt verschiedene Secret-Typen, die verwendet werden können, um Benutzernamen und Schlüssel im Secret-Objekt zu erzwingen: `service-account-token`, `basic-auth`, `ssh-auth`, `tls` und `opaque`. Der Standardtyp ist `opaque`. Der Typ `opaque` führt keine Validierung durch und ermöglicht unstrukturierte Schlüssel-Wert-Paare, die beliebige Werte enthalten können.

ConfigMap-Ressourcen ähneln Secret-Ressourcen, sie speichern jedoch keine sensiblen Daten. Mit einer ConfigMap-Ressource können sehr spezifische Informationen, wie persönliche Eigenschaften, oder allgemeine Informationen, wie die gesamten Konfigurationsdateien und JSON-Daten, gespeichert werden.

Sie können ConfigMap- und Secret-Ressourcen mit der OpenShift-CLI oder der Web Console erstellen. Anschließend können Sie in Ihrer Pod-Spezifikation auf sie verweisen. Woraufhin OpenShift automatisch die Ressourcendaten als Umgebungsvariablen in den Container oder als über Volumes bereitgestellte Dateien in den Anwendungscontainer einfügt.

Kapitel 2 | Entwerfen containerisierter Anwendungen für OpenShift

Sie können auch die Deploymentkonfiguration für eine aktive Anwendung ändern, um auf ConfigMap- und Secret-Ressourcen zu verweisen. OpenShift stellt die Anwendung anschließend erneut bereit und stellt die Daten dem Container zur Verfügung.

Daten werden in einer Secret-Ressource in der base64-Codierung gespeichert. Beim Einfügen der Daten aus einem Secret in einen Container werden die Daten decodiert und als Datei gemountet oder im Container als Umgebungsvariablen eingefügt.

Funktionen von Secrets und ConfigMaps

Beachten Sie die folgenden Punkte in Bezug auf Secrets und ConfigMaps:

- Sie können unabhängig von ihrer Definition referenziert werden.
- Gemountete Volumes für diese Ressourcen werden aus Sicherheitsgründen durch temporäre Dateispeicher (Temporary File Storage Facilities, tmpfs) unterstützt und niemals auf einem Knoten gespeichert.
- Sie gehören zu einem Namespace.

Erstellen und Verwalten von Secrets und ConfigMaps

Secrets und ConfigMaps müssen erstellt werden, bevor die von ihnen abhängigen Pods erstellt werden können. Sie können die CLI oder die Web Console verwenden, um diese Ressourcen zu erstellen.

Verwenden der Befehlszeile

Verwenden Sie den Befehl `oc create`, um Secrets und ConfigMap-Ressourcen zu erstellen.

So erstellen Sie eine neue ConfigMap, die Zeichenfolgenliterale speichert:

```
[user@host ~]$ oc create configmap config_map_name \
--from-literal key1=value1 \
--from-literal key2=value2
```

So erstellen Sie ein neues Secret, das Zeichenfolgenliterale speichert:

```
[user@host ~]$ oc create secret generic secret_name \
--from-literal username=user1 \
--from-literal password=mypa55w0rd
```

So erstellen Sie eine neue ConfigMap, die die Inhalte einer Datei oder eines Verzeichnisses, das einen Satz von Dateien enthält, speichert:

```
[user@host ~]$ oc create configmap config_map_name \
--from-file /home/demo/conf.txt
```

Wenn Sie eine ConfigMap anhand einer Datei erstellen, ist der Schlüsselname standardmäßig der Name der Datei und der Wert entspricht dem Inhalt der Datei.

Wenn Sie eine ConfigMap-Ressource auf Basis eines Verzeichnisses erstellen, wird jede Datei, deren Name ein gültiger Schlüssel im Verzeichnis ist, in der ConfigMap gespeichert. Unterverzeichnisse, symbolische Links, Gerätedateien und Pipes werden ignoriert.

Führen Sie den Befehl `oc create configmap --help` aus, um weitere Informationen zu erhalten.



Anmerkung

Sie können das Ressourcentypargument `configmap` auch als `cm` in der Befehlszeilschnittstelle `oc` abkürzen. Beispiel:

```
[user@host ~]$ oc create cm myconf --from-literal key1=value1
[user@host ~]$ oc get cm myconf
```

So erstellen Sie ein neues Secret, das den Inhalt einer Datei oder eines Verzeichnisses mit einem Satz von Dateien speichert:

```
[user@host ~]$ oc create secret generic secret_name \
--from-file /home/demo/mysecret.txt
```

Wenn Sie ein Secret anhand einer Datei oder eines Verzeichnisses erstellen, werden die Schlüsselnamen auf dieselbe Weise wie für ConfigMaps festgelegt.

Führen Sie die Befehle `oc create secret --help` und `oc secret` aus, um weitere Details, einschließlich zur Speicherung von TLS-Zertifikaten und Schlüsseln in Secrets zu erhalten.

Verwenden der OpenShift Web Console

Sie können auch die OpenShift Web Console verwenden, um ConfigMaps und Secrets zu erstellen. Melden Sie sich zum Erstellen und Verwalten von Secrets über die Web Console bei der OpenShift Web Console an und navigieren Sie zur Seite **Workloads → Secrets**.

Name	Namespace	Type	Size	Created	Actions
builder-dockercfg-tbf7z	your-project	kubernetes.io/dockercfg	1	Aug 3, 12:05 am	...
builder-token-cs2r4	your-project	kubernetes.io/service-account-token	4	Aug 3, 12:05 am	...
builder-token-hxj2h	your-project	kubernetes.io/service-account-token	4	Aug 3, 12:05 am	...
default-dockercfg-97qrn	your-project	kubernetes.io/dockercfg	1	Aug 3, 12:05 am	...
default-token-c892t	your-project	kubernetes.io/service-account-token	4	Aug 3, 12:05 am	...

Abbildung 2.2: Verwalten von Secrets über die Web Console

Navigieren Sie zum Erstellen und Verwalten von ConfigMaps über die Web Console zur Seite **Workloads → Config Maps**.

Name	Namespace	Size	Created
CM your-project-l-ca	NS your-project	1	Aug 3, 3:18 pm
CM your-project-l-global-ca	NS your-project	1	Aug 3, 3:18 pm
CM your-project-l-sys-config	NS your-project	0	Aug 3, 3:18 pm

Abbildung 2.3: Verwalten von ConfigMaps über die Web Console

Sie können den Wert, der jedem Schlüssel in einer ConfigMap zugewiesen ist, und auch den verschlüsselten Wert, der jedem Schlüssel in einem Secret zugewiesen ist, mit dem YAML-Editor der Web Console bearbeiten. Im Falle eines Secrets müssen Sie Ihre Daten jedoch im base64-Format codieren, bevor Sie sie in die Definition der Secret-Ressource einfügen können.

Definitionen von ConfigMaps und Secret-Ressourcen

Da ConfigMaps und Secrets reguläre OpenShift-Ressourcen sind, können Sie den Befehl `oc create` oder die Web Console verwenden, um diese Ressourcendefinitionsdateien im YAML- oder JSON-Format zu importieren.

Eine Beispieldefinition für ConfigMap-Ressourcen im YAML-Format ist unten dargestellt:

```
apiVersion: v1
data:
  key1: value1 ① ②
  key2: value2 ③ ④
kind: ConfigMap ⑤
metadata:
  name: myconf ⑥
```

- ① Der Name des ersten Schlüssels. Standardmäßig wird eine Umgebungsvariable oder eine Datei mit demselben Namen wie der Schlüssel in den Container eingefügt. Dies ist davon abhängig, ob die ConfigMap-Ressource als Umgebungsvariable oder Datei eingefügt wird.
- ② Der für den ersten Schlüssel der ConfigMap gespeicherte Wert.
- ③ Der Name des zweiten Schlüssels.
- ④ Der für den zweiten Schlüssel der ConfigMap gespeicherte Wert.
- ⑤ Der OpenShift-Ressourcentyp, in diesem Fall eine ConfigMap.
- ⑥ Ein eindeutiger Name für diese ConfigMap in einem Projekt.

Nachfolgend ist eine Beispiel-Secret-Ressource im YAML-Format dargestellt:

```
apiVersion: v1
data:
  username: cm9vdAo= 1 2
  password: c2VjcmV0Cg== 3 4
kind: Secret 5
metadata:
  name: mysecret 6
  type: Opaque
```

- 1** Der Name des ersten Schlüssels. Dadurch wird der Standardname für eine Umgebungsvariable oder eine Datei in einem Pod auf dieselbe Weise wie Schlüsselnamen aus der ConfigMap bereitgestellt.
- 2** Der für den ersten Schlüssel gespeicherte Wert im base64-codierten Format.
- 3** Der Name des zweiten Schlüssels.
- 4** Der für den zweiten Schlüssel gespeicherte Wert im base64-codierten Format.
- 5** Der OpenShift-Ressourcentyp, in diesem Fall ein Secret.
- 6** Ein eindeutiger Name für diese Secret-Ressource in einem Projekt.

Alternative Syntax für Secret-Ressourcen-Definitionen

In einer Vorlage können Secrets nicht mit der Standardsyntax definiert werden, da die Werte sämtlicher Schlüssel verschlüsselt sind. OpenShift stellt für dieses Szenario eine alternative Syntax bereit, wobei das Attribut `stringData` das Attribut `data` ersetzt und die Schlüsselwerte nicht verschlüsselt sind.

Mit der alternativen Syntax wird das vorherige Beispiel zu:

```
apiVersion: v1
stringData:
  username: user1
  password: pass1
kind: Secret
metadata:
  name: mysecret
  type: Opaque
```

Die alternative Syntax wird niemals in der etcd-Master-Datenbank von OpenShift gespeichert. OpenShift wandelt mit der alternativen Syntax definierte Secret-Ressourcen in die Standarddarstellung für den Storage um. Wenn Sie `oc get` mit einem Secret ausführen, das mit der alternativen Syntax erstellt wurde, erhalten Sie eine Ressource in der Standardsyntax.

Befehle zum Ändern von ConfigMaps

So zeigen Sie die Details einer ConfigMap im JSON-Format an oder exportieren eine ConfigMap-Ressourcedefinition in eine JSON-Datei für die Offline-Erstellung:

```
[user@host ~]$ oc get configmap/myconf -o json
```

So löschen Sie eine ConfigMap:

```
[user@host ~]$ oc delete configmap/myconf
```

Führen Sie zum Bearbeiten einer ConfigMap den Befehl `oc edit` aus. Dieser Befehl öffnet standardmäßig einen Vim-ähnlichen Puffer mit der ConfigMap-Ressourcendefinition im YAML-Format:

```
[user@host ~]$ oc edit configmap/myconf
```

Führen Sie den Befehl `oc patch` aus, um eine ConfigMap-Ressource zu bearbeiten. Dieser Ansatz ist nicht interaktiv und eignet sich, wenn Sie die Änderungen an einer Ressource in einem Skript erfassen müssen:

```
[user@host ~]$ oc patch configmap/myconf --patch '{"data":{"key1":"newValue1"}}'
```

Befehle zum Ändern von Secrets

Die Befehle zum Ändern von Secret-Ressourcen ähneln den Befehlen, die für ConfigMap-Ressourcen verwendet werden.

So zeigen Sie die Details eines Secrets an oder exportieren sie:

```
[user@host ~]$ oc get secret/mysecret -o json
```

So löschen Sie ein Secret:

```
[user@host ~]$ oc delete secret/mysecret
```

Zum Bearbeiten eines Secrets verschlüsseln Sie zunächst Ihre Daten im base64-Format, beispielsweise:

```
[user@host ~]$ echo 'newpassword' | base64  
bmV3cGFzc3dvcmQK
```

Verwenden Sie den verschlüsselten Wert, um die Secret-Ressource mit dem Befehl `oc edit` zu aktualisieren:

```
[user@host ~]$ oc edit secret/mysecret
```

Sie können eine Secret-Ressource auch mit dem Befehl `oc patch` bearbeiten:

```
[user@host ~]$ oc patch secret/mysecret --patch \  
'{"data":{"password":"bmV3cGFzc3dvcmQK"}}'
```

ConfigMaps und Secrets können auch mit der OpenShift Web Console geändert und gelöscht werden.

Einfügen von Daten aus Secrets und ConfigMaps in Anwendungen

ConfigMaps und Secrets können als Daten-Volumes gemountet oder als Umgebungsvariablen in einem Anwendungscontainer zur Verfügung gestellt werden.

Führen Sie den Befehl `oc set env` aus, um alle in einer ConfigMap gespeicherten Werte in Umgebungsvariablen für über eine Deploymentkonfiguration erstellte Pods einzufügen:

```
[user@host ~]$ oc set env deployment/mydcname \
--from configmap/myconf
```

Führen Sie den Befehl `oc set volume` aus, um alle Schlüssel aus einer ConfigMap als Dateien aus einem Volume in über eine Bereitstellung erstellte Pods zu mounten:

```
[user@host ~]$ oc set volume deployment/mydcname --add \
-t configmap -m /path/to/mount/volume \
--name myvol --configmap-name myconf
```

Führen Sie den Befehl `oc set env` aus, um Daten in einem Secret in Pods einzufügen, die aus einer Bereitstellung erstellt wurden:

```
[user@host ~]$ oc set env deployment/mydcname \
--from secret/mysecret
```

Führen Sie den Befehl `oc set volume` aus, um Daten von einer Secret-Ressource als Volume in Pods zu mounten, die aus einer Bereitstellung erstellt wurden:

```
[user@host ~]$ oc set volume deployment/mydcname --add \
-t secret -m /path/to/mount/volume \
--name myvol --secret-name mysecret
```

Anwendungskonfigurationsoptionen

Verwenden Sie für nicht sensible Informationen ConfigMaps zum Speichern von Konfigurationsdaten als reinen Text. Verwenden Sie Secrets, wenn die zu speichernden Informationen sensibel sind.

Wenn Ihre Anwendung nur ein paar einfache Konfigurationsvariablen aufweist, die von Umgebungsvariablen gelesen oder in der Befehlszeile übergeben werden können, sollten Sie Umgebungsvariablen verwenden, um Daten aus ConfigMaps und Secrets einzufügen. Umgebungsvariablen sind gegenüber dem Mounten von Volumes im Container der bevorzugte Ansatz.

Andererseits sollten Sie den Volume-Mount-Ansatz verwenden, wenn Ihre Anwendung eine große Anzahl an Konfigurationsvariablen aufweist oder wenn Sie eine Legacy-Anwendung migrieren, die Konfigurationsdateien umfassend einsetzt, anstatt eine Umgebungsvariable für jede Konfigurationsvariable zu erstellen. Wenn Ihre Anwendung beispielsweise eine oder mehrere Konfigurationsdateien von einem bestimmten Speicherort auf Ihrem Dateisystem erwartet, sollten Sie Secrets oder ConfigMaps aus den Konfigurationsdateien erstellen und sie im temporären Containerdateisystem an dem von der Anwendung erwarteten Speicherort mounten.

Kapitel 2 | Entwerfen containerisierter Anwendungen für OpenShift

Verwenden Sie den folgenden Befehl, um dieses Ziel zu erreichen, wobei Secrets auf die Datei /home/student/configuration.properties verweisen:

```
[user@host ~]$ oc create secret generic security \
--from-file /home/student/configuration.properties
```

Um das Secret in die Anwendung einzufügen, konfigurieren Sie ein Volume, das auf die im vorherigen Befehl erstellten Secrets verweist. Das Volume muss auf ein tatsächliches Verzeichnis innerhalb der Anwendung verweisen, in dem die Secrets-Datei gespeichert ist.

Im folgenden Beispiel wird die Datei configuration.properties im Verzeichnis /opt/app-root/secure gespeichert. Konfigurieren Sie die Deploymentkonfiguration aus der Anwendung (dc/application), um die Datei an die Anwendung zu binden:

```
[user@host ~]$ *oc set volume deployment/application --add \
-t secret -m /opt/app-root/secure \
--name myappsec-vol --secret-name security *
```

Verwenden Sie den folgenden Befehl, um eine ConfigMap zu erstellen:

```
[user@host ~]$ oc create configmap properties \
--from-file /home/student/configuration.properties
```

Zum Binden der Anwendung an die ConfigMap aktualisieren Sie die Deploymentkonfiguration aus dieser Anwendung so, dass die ConfigMap verwendet wird:

```
[user@host ~]$ oc set env deployment/application \
--from configmap/properties
```



Literaturhinweise

Weitere Informationen zu Secrets finden Sie im Kapitel *Providing Sensitive Data to Pods* des Handbuchs *Nodes* für Red Hat OpenShift Container Platform 4.6 unter https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/nodes/working-with-pods#nodes-pods-secrets

► Angeleitete Übung

Einfügen von Konfigurationsdaten in eine Anwendung

In dieser Übung verwenden Sie ConfigMaps und Secrets, um die Konfiguration für eine containerisierte Anwendung zu externalisieren.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Bereitstellen einer einfachen Node.js-basierten Anwendung, die Konfigurationsdetails aus Umgebungsvariablen und Dateien ausgibt
- Einfügen von Konfigurationsdaten in den Container mit ConfigMaps und Secrets
- Ändern der Daten in der ConfigMap und Verifizieren, ob die Anwendung die geänderten Werte auswählt

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen aktiven OpenShift-Cluster
- Auf das S2I-Builder-Image für Node.js 12
- Auf die Beispielanwendung im Git-Repository (`app-config`)

Führen Sie den folgenden Befehl auf der `workstation`-VM aus, um die Voraussetzungen für die Übung zu überprüfen sowie die Übungs- und Lösungsdateien herunterzuladen:

```
[student@workstation ~]$ lab app-config start
```

Anweisungen

► 1. Überprüfen Sie den Anwendungsquellcode.

- 1.1. Wechseln Sie zu Ihrem lokalen Klon des D0288-apps-Git-Repository und checken Sie den `main`-Branch des Kurs-Repository aus, um sicherzustellen, dass Sie diese Übung von einem als fehlerfrei bekannten Zustand aus starten:

```
[student@workstation ~]$ cd D0288-apps
[student@workstation D0288-apps]$ git checkout main
...output omitted...
```

- 1.2. Erstellen Sie einen neuen Branch, in dem Sie alle Änderungen speichern können, die Sie während dieser Übung vornehmen:

```
[student@workstation D0288-apps]$ git checkout -b app-config
Switched to a new branch 'app-config'
[student@workstation D0288-apps]$ git push -u origin app-config
...output omitted...
* [new branch]      app-config -> app-config
Branch app-config set up to track remote branch app-config from origin.
```

- 1.3. Überprüfen Sie die Datei /home/student/D0288-apps/app-config/app.js.

Die Anwendung liest den Wert der Umgebungsvariablen APP_MSG und druckt die Inhalte der Datei /opt/app-root/secure/myapp.sec:

```
// read in the APP_MSG env var
var msg = process.env.APP_MSG;
...output omitted...
// Read in the secret file
fs.readFile('/opt/app-root/secure/myapp.sec', 'utf8', function (secerr, secdata) {
...output omitted...
```

► 2. Erzeugen Sie die Anwendung und stellen Sie sie bereit.

- 2.1. Laden Sie die Konfiguration Ihrer Kursumgebung.

Führen Sie den folgenden Befehl aus, um die in der ersten angeleiteten Übung erstellten Umgebungsvariablen zu laden:

```
[student@workstation D0288-apps]$ source /usr/local/etc/ocp4.config
```

- 2.2. Melden Sie sich bei OpenShift mit Ihrem Entwicklerbenutzerkonto an:

```
[student@workstation D0288-apps]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful
...output omitted...
```

- 2.3. Erstellen Sie ein neues Projekt für die Anwendung. Stellen Sie dem Projektnamen Ihren Entwicklerbenutzernamen voran:

```
[student@workstation D0288-apps]$ oc new-project ${RHT_OCP4_DEV_USER}-app-config
```

- 2.4. Erstellen Sie eine neue Anwendung mit dem Namen myapp anhand von Quellen in Git. Verwenden Sie den Branch, den Sie in einem vorherigen Schritt erstellt haben.

Sie können den Befehl aus dem Skript oc-new-app.sh im Ordner /home/student/D0288/labs/app-config kopieren oder ausführen:

```
[student@workstation D0288-apps]$ oc new-app --name myapp --build-env \
npm_config_registry=http://${RHT_OCP4_NEXUS_SERVER}/repository/nodejs \
nodejs:12~https://github.com/${RHT_OCP4_GITHUB_USER}/D0288-apps#app-config \
--context-dir app-config
...output omitted...
--> Creating resources ...output omitted...
```

```
imagestream.image.openshift.io "myapp" created
buildconfig.build.openshift.io "myapp" created
deployment.apps.openshift.io "myapp" created
service "myapp" created
--> Success
...output omitted...
```

Beachten Sie, dass vor und hinter dem Gleichheitszeichen (=) nach `npm_config_registry` kein Leerzeichen steht.

- 2.5. Sehen Sie sich die Build-Logs an. Warten Sie, bis der Build abgeschlossen ist und das Anwendungscontainer-Image an die interne OpenShift-Registry übertragen wurde:

```
[student@workstation D0288-apps]$ oc logs -f bc/myapp
Cloning "https://github.com/youruser/D0288-apps#app-config" ...
...output omitted...
--> Installing application source ...
--> Building your Node application from source
...output omitted...
Pushing image image-registry.openshift-image-registry.svc:5000/youruser-app-
config/myapp:latest ...
...output omitted...
Push successful
```

► 3. Testen Sie die Anwendung.

- 3.1. Warten Sie, bis die Anwendung bereitgestellt wird. Zeigen Sie den Status des Anwendungs-Pods an. Der Anwendungs-Pod sollte den Status Running aufweisen:

```
[student@workstation D0288-apps]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
myapp-1-build  0/1     Completed  0          65s
myapp-597fdb8cc9-z6t86  1/1     Running   0          29s
```

- 3.2. Verwenden Sie eine Route, um die Anwendung für den externen Zugriff bereitzustellen:

```
[student@workstation D0288-apps]$ oc expose svc myapp
route.route.openshift.io/myapp exposed
```

- 3.3. Identifizieren Sie die Routen-URL, für die die Anwendungs-API verfügbar gemacht wird:

```
[student@workstation D0288-apps]$ oc get route
NAME      HOST/PORT           ...
myapp    myapp-youruser-app-config.apps.cluster.domain.example.com ...
```

- 3.4. Rufen Sie die Routen-URL auf, die im vorherigen Schritt mit dem Befehl `curl` identifiziert wurde:

```
[student@workstation D0288-apps]$ curl \
http://myapp-${RHT_OCP4_DEV_USER}-app-config.${RHT_OCP4_WILDCARD_DOMAIN}
Value in the APP_MSG env var is => undefined
Error: ENOENT: no such file or directory, open '/opt/app-root/secure/myapp.sec'
```

Es werden der Wert `undefined` für die Umgebungsvariable und der Fehler `ENOENT: no such file or directory` angezeigt, da keine solche Umgebungsvariable oder Datei im Container vorhanden ist.

► 4. Erstellen Sie die ConfigMap- und Secret-Ressourcen.

- 4.1. Erstellen Sie eine ConfigMap-Ressource zum Aufnehmen von Konfigurationsvariablen, die Daten als reinen Text speichern.

Erstellen Sie eine neue ConfigMap-Ressource mit dem Namen `myappconf`. Speichern Sie einen Schlüssel mit dem Namen `APP_MSG` und dem Wert `Test Message` in dieser ConfigMap:

```
[student@workstation D0288-apps]$ oc create configmap myappconf \
--from-literal APP_MSG="Test Message"
configmap/myappconf created
```

- 4.2. Überprüfen Sie, ob die ConfigMap die Konfigurationsdaten enthält:

```
[student@workstation D0288-apps]$ oc describe cm/myappconf
Name: myappconf
...output omitted...
Data
=====
APP_MSG:
---
Test Message
...output omitted...
```

- 4.3. Überprüfen Sie den Inhalt der Datei `/home/student/D0288-apps/app-config/myapp.sec`:

```
username=user1
password=pass1
salt=xyz123
```

- 4.4. Erstellen Sie ein neues Secret, um die Inhalte der Datei `myapp.sec` zu speichern.

```
[student@workstation D0288-apps]$ oc create secret generic myappfilesec \
--from-file /home/student/D0288-apps/app-config/myapp.sec
secret/myappfilesec created
```

- 4.5. Verifizieren Sie die Inhalte des Secrets. Beachten Sie, dass die Inhalte im base64-codierten Format gespeichert werden:

```
[student@workstation D0288-apps]$ oc get secret/myappfilesec -o json
{
  "apiVersion": "v1",
  "data": {
    "myapp.sec": "dXNlc5hbWU9dXNlcjEKcGFzc3dvcmQ9cGFzcxEKc2...
  },
  "kind": "Secret",
  "metadata": {
    ...output omitted...
    "name": "myappfilesec",
    ...output omitted...
  },
  "type": "Opaque"
}
```

► 5. Fügen Sie die ConfigMap und das Secret in den Anwendungscontainer ein.

- 5.1. Führen Sie den Befehl `oc set env` aus, um der Deploymentkonfiguration die ConfigMap hinzuzufügen:

```
[student@workstation ~]$ oc set env deployment/myapp \
--from configmap/myappconf
deployment.apps.openshift.io/myapp updated
```

- 5.2. Führen Sie den Befehl `oc set volume` aus, um der Deploymentkonfiguration das Secret hinzuzufügen:

Sie können den Befehl aus dem Skript `inject-secret-file.sh` im Ordner / home/student/D0288/labs/app-config kopieren oder ausführen:

```
[student@workstation D0288-apps]$ oc set volume deployment/myapp --add \
-t secret -m /opt/app-root/secure \
--name myappsec-vol --secret-name myappfilesec
deployment.apps/myapp volume updated
```

► 6. Verifizieren Sie, ob die Anwendung erneut bereitgestellt wird und die Daten aus der ConfigMap und aus dem Secret verwendet.

- 6.1. Verifizieren Sie, ob die Anwendung aufgrund der in den vorherigen Schritten an der Bereitstellung vorgenommenen Änderungen erneut bereitgestellt wird:

```
[student@workstation D0288-apps]$ oc status
In project youruser-app-config on server ...output omitted...

http://myapp-youruser-app-config.apps.cluster.domain.example.com to pod port 8080-
tcp (svc/myapp)
deployment/myapp deploys istag/myapp:latest <-
bc/myapp source builds https://github.com/youruser/D0288-apps#app-config on
openshift/nodejs:12
deployment #4 running for 10 seconds - 1 pod
```

```
deployment #3 deployed 44 seconds ago
deployment #2 deployed 3 minutes ago
deployment #1 deployed 4 minutes ago
```

**Anmerkung**

Die Anwendung sollte aufgrund der zwei `oc set env`-Befehle, die die Bereitstellung ändern, zweimal erneut bereitgestellt werden.

Sie können Fehlermeldungen wie die folgende problemlos ignorieren:

```
deployment #2 failed 59 seconds ago: newer deployment was found running
```

- 6.2. Warten Sie, bis der Anwendungs-Pod bereit ist und sein Zustand `Running` lautet. Rufen Sie mit dem Befehl `oc get pods` den Namen des Anwendungs-Pods ab.

```
[student@workstation D0288-apps]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
myapp-1-build  0/1     Completed  0          8m12s
myapp-ddffbc7f9-ntsjq  1/1     Running   0          3m53s
```

- 6.3. Führen Sie den Befehl `oc rsh` aus, um die Umgebungsvariablen im Container zu überprüfen:

```
[student@workstation D0288-apps]$ oc rsh myapp-ddffbc7f9-ntsjq env | grep APP_MSG
APP_MSG=Test Message
```

- 6.4. Überprüfen Sie, ob die ConfigMap und das Secret in den Container eingefügt wurden. Testen Sie die Anwendung erneut mit der Routen-URL:

```
[student@workstation D0288-apps]$ curl \
http://myapp-${RHT_OCP4_DEV_USER}-app-config.${RHT_OCP4_WILDCARD_DOMAIN}
Value in the APP_MSG env var is => Test Message
The secret is => username=user1
password=pass1
salt=xyz123
```

OpenShift fügt die ConfigMap als Umgebungsvariable ein und stellt das Secret im Container als Datei bereit. Die Anwendung liest die Umgebungsvariable und die Datei und zeigt ihre Daten an.

- 7. Ändern Sie die in der ConfigMap gespeicherten Informationen und testen Sie die Anwendung erneut.

- 7.1. Führen Sie den Befehl `oc edit configmap` aus, um den Wert des Schlüssels `APP_MSG` zu ändern:

```
[student@workstation D0288-apps]$ oc edit cm/myappconf
```

Durch den obigen Befehl wird ein Vim-ähnlicher Puffer mit den ConfigMap-Attributen im YAML-Format geöffnet. Ändern Sie den Wert, der dem Schlüssel `APP_MSG` im Abschnitt „`data`“ zugeordnet ist, wie folgt:

```
...output omitted...
apiVersion: v1
data:
  APP_MSG: Changed Test Message
kind: ConfigMap
...output omitted...
```

Speichern und schließen Sie die Datei.

- 7.2. Verifizieren Sie, ob der Wert im Schlüssel APP_MSG aktualisiert wird:

```
[student@workstation D0288-apps]$ oc describe cm/myappconf
Name: myappconf
...output omitted...
Data
=====
APP_MSG:
---
Changed Test Message
...output omitted...
```

- 7.3. Führen Sie den Befehl `oc delete pod` aus, um eine neue Bereitstellung auszulösen. Nach dem Löschen des Pods stellt der Replication Controller automatisch einen neuen Pod zur Verfügung. Dadurch wird gewährleistet, dass die Anwendung die geänderten Werte in der ConfigMap übernimmt:

```
[student@workstation D0288-apps]$ oc delete pod myapp-ddffbc7f9-ntsjq
pod "myapp-ddffbc7f9-ntsjq" deleted
```

- 7.4. Warten Sie, bis der Anwendungs-Pod erneut bereitgestellt und im Zustand `Running` angezeigt wird.

```
[student@workstation D0288-apps]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
myapp-1-build  0/1     Completed  0          16m
myapp-ddffbc7f9-5wls6  1/1     Running   0          2m37s
```

- 7.5. Testen Sie die Anwendung und überprüfen Sie, ob die geänderten Werte in der ConfigMap angezeigt werden:

```
[student@workstation D0288-apps]$ curl \
  http://myapp-${RHT_OCP4_DEV_USER}-app-config.${RHT_OCP4_WILDCARD_DOMAIN}
Value in the APP_MSG env var is => Changed Test Message
The secret is => username=user1
password=pass1
salt=xyz123
```

- 8. Führen Sie eine Bereinigung durch. Löschen Sie das Projekt `youruser-app-config` in OpenShift.

```
[student@workstation D0288-apps]$ oc delete project \
${RHT_OCP4_DEV_USER}-app-config
project.project.openshift.io "youruser-app-config" deleted
```

Beenden

Führen Sie auf der `workstation` den Befehl `lab app-config finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken. Die Aktion „Beenden“ veröffentlicht dieses Projekt und seine Ressourcen.

```
[student@workstation D0288-apps]$ lab app-config finish
```

Hiermit ist die angeleitete Übung beendet.

► Praktische Übung

Entwerfen containerisierter Anwendungen für OpenShift

In dieser praktischen Übung korrigieren Sie das Containerfile, um eine auf dem Thorntail-Framework basierende Anwendung in einem OpenShift-Cluster auszuführen. Sie verwenden zudem eine ConfigMap zum Konfigurieren der Anwendung.



Anmerkung

Für den Befehl grade am Ende jeder praktischen Übung eines Kapitels ist es erforderlich, dass Sie die exakten Projektnamen und anderen Identifikatoren, wie in der Spezifikation der praktischen Übung angegeben, verwenden.

Ergebnisse

Sie sollten in der Lage sein, das Containerfile für eine auf dem Thorntail-Framework basierende Anwendung so zu korrigieren, dass sie als zufälliger Benutzer ausgeführt wird. Zudem sollten Sie die Anwendung in einem OpenShift-Cluster bereitstellen können. Sie sollten außerdem in der Lage sein, mit einer ConfigMap eine einfache Textzeichenfolge zu speichern, die zum Konfigurieren der Anwendung verwendet wird.

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen aktiven OpenShift-Cluster
- Auf die ausführbare Fat-JAR der Anwendung
- Auf das Git-Repository der Anwendung

Führen Sie den folgenden Befehl auf `workstation` aus, um die Voraussetzungen zu überprüfen. Der Befehl lädt auch die Hilfs- und Lösungsdateien für die praktische Übung herunter:

```
[student@workstation ~]$ lab design-container start
```

Anforderungen

Die Anwendung wird in Java unter Verwendung des Thorntail-Frameworks geschrieben. Es werden die vorab erstellte ausführbare JAR-Datei (Fat-JAR) mit enthaltener Anwendung und die Thorntail-Laufzeit bereitgestellt. Die Anwendung enthält eine einfache REST-API, die anhand einer in den Container als Umgebungsvariable eingefügte Konfiguration auf Anforderungen antwortet. Erstellen Sie die Anwendung entsprechend den folgenden Anforderungen, und stellen Sie sie auf einem OpenShift-Cluster bereit:

- Der Anwendungsname für OpenShift lautet `elvis`. Die Konfiguration für die Anwendung sollte in einer ConfigMap mit dem Namen `appconfig` gespeichert werden.
- Stellen Sie die Anwendung in einem Projekt mit dem Namen `youruser-design-container` bereit.

- Der Zugriff auf die REST-API für die Anwendung sollte unter der folgenden URL möglich sein:
`elvis-youruser-design-container.apps.cluster.domain.example.com/api/hello.`
- Das Git-Repository und der Git-Ordner, der die Anwendungsquellen enthält, lautet:
`https://github.com/youruser/D0288-apps/hello-java.`
- Die vorab erstellte Anwendungs-JAR-Datei ist verfügbar unter:
`https://github.com/RedHatTraining/D0288-apps/releases/download/OCP-4.1-1/hello-java.jar`

Anweisungen

- Navigieren Sie zu Ihrem lokalen Klon des D0288-apps-Git-Repository und erstellen Sie aus dem main-Branch einen neuen Branch mit dem Namen `design-container`. Überprüfen Sie kurz das Containerfile für die Anwendung im Verzeichnis `/home/student/D0288-apps/hello-java/`.
- Stellen Sie die Anwendung im Ordner `hello-java` des D0288-apps-Git-Repository mithilfe des `design-container`-Branches der Anwendung bereit. Stellen Sie die Anwendung im Projekt `youruser-design-container` in OpenShift bereit, ohne Änderungen daran vorzunehmen.
Vergessen Sie nicht, die Variablen in der Datei `/usr/local/etc/ocp4.config` zu speichern, bevor Sie sich beim OpenShift-Cluster anmelden.
- Zeigen Sie den Deploymentstatus des Anwendungs-Pods an. Der Pod weist den Zustand `CrashLoopBackOff` oder `Error` auf. Zeigen Sie die AnwendungsLogs an, um festzustellen, warum die Anwendung nicht ordnungsgemäß gestartet wird.
- Bearbeiten Sie das Containerfile für die Anwendung, um die erfolgreiche Bereitstellung in einem OpenShift-Cluster zu gewährleisten. Der Container sollte mit einer zufälligen Benutzer-ID und nicht mit dem aktuell konfigurierten Benutzer `wildfly` ausgeführt werden.
- Übergeben Sie die am Containerfile vorgenommenen Änderungen und übertragen Sie die Änderungen in das Git-Repository des Kursraums.
- Starten Sie einen neuen Build der Anwendung. Beobachten Sie das Build-Log für den neuen Build. Vergewissern Sie sich, dass der Anwendungs-Pod erfolgreich gestartet wird.
- Stellen Sie den Service für den externen Zugriff bereit und testen Sie die Anwendung. Greifen Sie über den Kontextpfad `/api/hello` auf die API der Anwendung zu.
- Erstellen Sie eine neue ConfigMap mit dem Namen `appconfig`. Speichern Sie einen Schlüssel mit dem Namen `APP_MSG` mit dem Wert `Elvis lives` in dieser ConfigMap. Fügen Sie diesen Schlüssel als Umgebungsvariable zur Deploymentkonfiguration der Anwendung hinzu.
- Überprüfen Sie, ob eine neue Bereitstellung ausgelöst wird, und warten Sie, bis der neue Anwendungs-Pod bereit ist und ausgeführt wird. Verifizieren Sie, ob der Schlüssel `APP_MSG` als eine Umgebungsvariable in den Container injiziert wird.
- Testen Sie die Anwendung, indem Sie deren REST-API-URL (`http://elvis-youruser-design-container.apps.cluster.domain.example.com/api/hello`) aufrufen, und vergewissern Sie sich, dass der Schlüsselwert `APP_MSG` in der Antwort erscheint.

Bewertung

Verwenden Sie als Benutzer `student` auf dem Rechner `workstation` den Befehl `lab`, um Ihre Arbeit zu bewerten. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie den Befehl so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab design-container grade
```

Beenden

Führen Sie auf der `workstation` den Befehl `lab design-container finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken. Die Aktion „Beenden“ veröffentlicht dieses Projekt und seine Ressourcen.

```
[student@workstation ~]$ lab design-container finish
```

Hiermit ist die praktische Übung abgeschlossen.

► Lösung

Entwerfen containerisierter Anwendungen für OpenShift

In dieser praktischen Übung korrigieren Sie das Containerfile, um eine auf dem Thorntail-Framework basierende Anwendung in einem OpenShift-Cluster auszuführen. Sie verwenden zudem eine ConfigMap zum Konfigurieren der Anwendung.



Anmerkung

Für den Befehl `grade` am Ende jeder praktischen Übung eines Kapitels ist es erforderlich, dass Sie die exakten Projektnamen und anderen Identifikatoren, wie in der Spezifikation der praktischen Übung angegeben, verwenden.

Ergebnisse

Sie sollten in der Lage sein, das Containerfile für eine auf dem Thorntail-Framework basierende Anwendung so zu korrigieren, dass sie als zufälliger Benutzer ausgeführt wird. Zudem sollten Sie die Anwendung in einem OpenShift-Cluster bereitstellen können. Sie sollten außerdem in der Lage sein, mit einer ConfigMap eine einfache Textzeichenfolge zu speichern, die zum Konfigurieren der Anwendung verwendet wird.

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen aktiven OpenShift-Cluster
- Auf die ausführbare Fat-JAR der Anwendung
- Auf das Git-Repository der Anwendung

Führen Sie den folgenden Befehl auf `workstation` aus, um die Voraussetzungen zu überprüfen. Der Befehl lädt auch die Hilfs- und Lösungsdateien für die praktische Übung herunter:

```
[student@workstation ~]$ lab design-container start
```

Anforderungen

Die Anwendung wird in Java unter Verwendung des Thorntail-Frameworks geschrieben. Es werden die vorab erstellte ausführbare JAR-Datei (Fat-JAR) mit enthaltener Anwendung und die Thorntail-Laufzeit bereitgestellt. Die Anwendung enthält eine einfache REST-API, die anhand einer in den Container als Umgebungsvariable eingefügte Konfiguration auf Anforderungen antwortet. Erstellen Sie die Anwendung entsprechend den folgenden Anforderungen, und stellen Sie sie auf einem OpenShift-Cluster bereit:

- Der Anwendungsname für OpenShift lautet `elvis`. Die Konfiguration für die Anwendung sollte in einer ConfigMap mit dem Namen `appconfig` gespeichert werden.
- Stellen Sie die Anwendung in einem Projekt mit dem Namen `youruser-design-container` bereit.

- Der Zugriff auf die REST-API für die Anwendung sollte unter der folgenden URL möglich sein:
`elvis-youruser-design-container.apps.cluster.domain.example.com/api/hello.`
- Das Git-Repository und der Git-Ordner, der die Anwendungsquellen enthält, lautet:
`https://github.com/youruser/D0288-apps/hello-java.`
- Die vorab erstellte Anwendungs-JAR-Datei ist verfügbar unter:
`https://github.com/RedHatTraining/D0288-apps/releases/download/OCP-4.1-1/hello-java.jar`

Anweisungen

- Navigieren Sie zu Ihrem lokalen Klon des D0288-apps-Git-Repository und erstellen Sie aus dem main-Branch einen neuen Branch mit dem Namen `design-container`. Überprüfen Sie kurz das Containerfile für die Anwendung im Verzeichnis `/home/student/D0288-apps/hello-java/`.
 - Checken Sie den main-Branch des Git-Repository aus.

```
[student@workstation ~]$ cd D0288-apps
[student@workstation D0288-apps]$ git checkout main
...output omitted...
```

- Erstellen Sie einen neuen Branch, in dem Sie alle Änderungen speichern können, die Sie während dieser Übung vornehmen:

```
[student@workstation D0288-apps]$ git checkout -b design-container
Switched to a new branch 'design-container'
[student@workstation D0288-apps]$ git push -u origin design-container
...output omitted...
* [new branch]      design-container -> design-container
Branch design-container set up to track remote branch design-container from
origin.
```
- Überprüfen Sie die Datei `/home/student/D0288-apps/hello-java/Containerfile`. Nehmen Sie daran nun keine Änderungen vor.

- Stellen Sie die Anwendung im Ordner `hello-java` des D0288-apps-Git-Repository mithilfe des `design-container`-Branches der Anwendung bereit. Stellen Sie die Anwendung im Projekt `youruser-design-container` in OpenShift bereit, ohne Änderungen daran vorzunehmen.
Vergessen Sie nicht, die Variablen in der Datei `/usr/local/etc/ocp4.config` zu speichern, bevor Sie sich beim OpenShift-Cluster anmelden.

- 2.1. Laden Sie die Konfiguration Ihrer Kursumgebung.

Führen Sie den folgenden Befehl aus, um die in der ersten angeleiteten Übung erstellten Konfigurationsvariablen zu laden:

```
[student@workstation D0288-apps]$ source /usr/local/etc/ocp4.config
```

- 2.2. Melden Sie sich bei OpenShift mit Ihrem Entwicklerbenutzerkonto an:

```
[student@workstation D0288-apps]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful
...output omitted...
```

- 2.3. Erstellen Sie ein neues Projekt für die Anwendung. Stellen Sie dem Namen des Projekts den Entwicklerbenutzernamen voran:

```
[student@workstation D0288-apps]$ oc new-project \
${RHT_OCP4_DEV_USER}-design-container
```

- 2.4. Erstellen Sie ein neues Image aus dem Containerfile.

```
[student@workstation D0288-apps]$ podman build --layers=false \
-t do288-hello-java ./hello-java
STEP 1: FROM registry.access.redhat.com/ubi8/ubi:8.0
...output omitted...
STEP 11: COMMIT do288-hello-java
...output omitted...
```

- 2.5. Versehen Sie das neue Image mit einem Tag, und übertragen Sie es an Quay.io.

```
[student@workstation D0288-apps]$ podman tag do288-hello-java \
quay.io/${RHT_OCP4_QUAY_USER}/do288-hello-java
[student@workstation D0288-apps]$ podman login quay.io -u ${RHT_OCP4_QUAY_USER}
Password:
Login Succeeded!
[student@workstation D0288-apps]$ podman push \
quay.io/${RHT_OCP4_QUAY_USER}/do288-hello-java
...output omitted...
Storing signatures
```

- 2.6. Melden Sie sich bei Quay.io [http://quay.io] an, und veröffentlichen Sie das neu hinzugefügte Image. Der Befehl „new-app“ schlägt ohne diesen Schritt fehl.

- 2.7. Verwenden Sie das Image im Repository, um eine neue Anwendung in OpenShift mit dem Namen elvis zu erstellen.

```
[student@workstation D0288-apps]$ oc new-app --name elvis \
quay.io/${RHT_OCP4_QUAY_USER}/do288-hello-java
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "elvis" created
deployment.apps "elvis" created
service "elvis" created
--> Success
...output omitted...
```

3. Zeigen Sie den Deploymentstatus des Anwendungs-Pods an. Der Pod weist den Zustand CrashLoopBackOff oder Error auf. Zeigen Sie die AnwendungsLogs an, um festzustellen, warum die Anwendung nicht ordnungsgemäß gestartet wird.
 - 3.1. Warten Sie, bis der Anwendungs-Pod bereitgestellt ist. Die Anwendung erreicht nicht den Status Ready. Es verbleibt nach einiger Zeit entweder im Status CrashLoopBackOff oder im Status Error.

NAME	READY	STATUS	RESTARTS	AGE
elvis-799f8df69b-vh6fr	0/1	Error	2	34s

- 3.2. Zeigen Sie die Logs für den Anwendungs-Pod an und untersuchen Sie, weshalb der Anwendungs-Pod nicht gestartet werden konnte:

```
[student@workstation D0288-apps]$ oc logs elvis-799f8df69b-vh6fr
/bin/sh: /opt/app-root/bin/run-app.sh: Permission denied
```

Die Anwendung kann aufgrund des Fehlers „Permission denied“ im Dateisystem nicht gestartet werden, da OpenShift den Pod nicht mit dem in dem Containerfile angegebenen Benutzer ausführt.

4. Bearbeiten Sie das Containerfile für die Anwendung, um die erfolgreiche Bereitstellung in einem OpenShift-Cluster zu gewährleisten. Der Container sollte mit einer zufälligen Benutzer-ID und nicht mit dem aktuell konfigurierten Benutzer wildfly ausgeführt werden.
 - 4.1. Bearbeiten Sie das Containerfile unter /home/student/D0288-apps/hello-java/Containerfile mit einem Texteditor. Nehmen Sie die in den folgenden Schritten beschriebenen Änderungen vor. Sie können auch die Anweisungen und Befehle aus der unter /home/student/D0288/solutions/design-container/Containerfile bereitgestellten Lösungsdatei kopieren.
Entfernen Sie den Befehl useradd in der ersten RUN-Anweisung (Zeile 12).

```
useradd wildfly && \
```

- 4.2. Suchen Sie die Befehle chown und chmod in den Zeilen 19 und 20:

```
RUN chown -R wildfly:wildfly /opt/app-root && \
chmod -R 700 /opt/app-root
```

Ersetzen Sie sie durch Folgendes:

```
RUN chgrp -R 0 /opt/app-root && \
    chmod -R g=u /opt/app-root
```

- 4.3. Ersetzen Sie den Benutzer `wildfly` in der `USER`-Anweisung in Zeile 24 durch die generische „`userid`“ 1001, um zu verhindern, dass der Benutzer aus dem übergeordneten RHEL-Image übernommen wird. Diese generische „`userid`“ wird von OpenShift ignoriert und folgt den Red Hat-Empfehlungen und -Konventionen für das Entwickeln von Images:

```
USER 1001
```

5. Übergeben Sie die am Containerfile vorgenommenen Änderungen und übertragen Sie die Änderungen in das Git-Repository des Kursraums.

```
[student@workstation D0288-apps]$ cd hello-java
[student@workstation hello-java]$ git commit -a -m \
"Fixed Containerfile to run with random user id on OpenShift"
[student@workstation hello-java]$ git push
[student@workstation hello-java]$ cd ..
[student@workstation D0288-apps]$
```

6. Starten Sie einen neuen Build der Anwendung. Beobachten Sie das Build-Log für den neuen Build. Vergewissern Sie sich, dass der Anwendungs-Pod erfolgreich gestartet wird.

- 6.1. Starten Sie einen neuen Build für die Anwendung:

```
[student@workstation D0288-apps]$ podman rmi -a --force
...output omitted...
[student@workstation D0288-apps]$ podman build --layers=false \
-t do288-hello-java ./hello-java
STEP 1: FROM registry.access.redhat.com/ubi8/ubi:8.0
...output omitted...
STEP 7: RUN chgrp -R 0 /opt/app-root &&           chmod -R g=u /opt/app-root
...output omitted...
STEP 9: USER 1001
...output omitted...
```

- 6.2. Versehen Sie das neue Image mit einem Tag, und übertragen Sie es an Quay.io.

```
[student@workstation D0288-apps]$ podman tag do288-hello-java \
quay.io/${RHT_OCP4_QUAY_USER}/do288-hello-java
[student@workstation D0288-apps]$ podman push \
quay.io/${RHT_OCP4_QUAY_USER}/do288-hello-java
...output omitted...
Storing signatures
```

- 6.3. Entfernen Sie das alte Projekt, und erstellen Sie es neu.

```
[student@workstation D0288-apps]$ oc delete project \
${RHT_OCP4_DEV_USER}-design-container
...output omitted...
[student@workstation D0288-apps]$ oc new-project \
${RHT_OCP4_DEV_USER}-design-container
```

- 6.4. Verwenden Sie das aktualisierte Image im Repository, um eine neue Anwendung im OpenShift-Cluster elvis zu erstellen.

```
[student@workstation D0288-apps]$ oc new-app --name elvis \
quay.io/${RHT_OCP4_QUAY_USER}/do288-hello-java
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "elvis" created
deployment.apps "elvis" created
service "elvis" created
--> Success
...output omitted...
```

- 6.5. Warten Sie, bis der Anwendungs-Pod bereit ist und ausgeführt wird.

```
[student@workstation D0288-apps]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
elvis-6d4fc74867-b6z2h   1/1     Running   0          29s
```

- 6.6. Zeigen Sie die Logs für den Anwendungs-Pod an und vergewissern Sie sich, dass während des Starts keine Fehler auftreten:

```
[student@workstation ~]$ oc logs elvis-6d4fc74867-b6z2h
Starting hello-java app...
JVM options => -Xmx512m
...output omitted...
2021-06-22 17:43:56,211 INFO  [org.wildfly.extension.undertow] (MSC service thread
1-2) WFLYUT0018: Host default-host starting
...output omitted...
2021-06-22 17:43:56,484 INFO  [org.wildfly.swarm] (main) THORN99999: Thorntail is
Ready
```

7. Stellen Sie den Service für den externen Zugriff bereit und testen Sie die Anwendung. Greifen Sie über den Kontextpfad /api/hello auf die API der Anwendung zu.

- 7.1. Stellen Sie die Anwendung für den externen Zugriff bereit.

```
[student@workstation ~]$ oc expose svc/elvis
route.route.openshift.io/elvis exposed
```

- 7.2. Identifizieren Sie den Hostnamen, für den die Anwendungs-API verfügbar gemacht wird:

```
[student@workstation ~]$ oc get route  
NAME      HOST/PORT  
elvis     elvis-youruser-design-container.apps.cluster.domain.example.com
```

- 7.3. Testen Sie die Anwendung, indem Sie die API-URL mit dem im vorherigen Schritt identifizierten Hostnamen aufrufen (<http://elvis-youruser-design-container.apps.cluster.domain.example.com/api/hello>), und vergewissern Sie sich, dass der Anwendungs-Pod-Name in der Antwort erscheint:

```
[student@workstation ~]$ curl \  
http://elvis-${RHT_OCP4_DEV_USER}-design-container.${RHT_OCP4_WILDCARD_DOMAIN}\ \  
/api/hello  
Hello world from host elvis-6d4fc74867-b6z2h
```

8. Erstellen Sie eine neue ConfigMap mit dem Namen appconfig. Speichern Sie einen Schlüssel mit dem Namen APP_MSG mit dem Wert Elvis lives in dieser ConfigMap. Fügen Sie diesen Schlüssel als Umgebungsvariable zur Deploymentkonfiguration der Anwendung hinzu.

- 8.1. Erstellen Sie die ConfigMap:

```
[student@workstation ~]$ oc create cm appconfig \  
--from-literal APP_MSG="Elvis lives"  
configmap/appconfig created
```

- 8.2. Zeigen Sie die Details der ConfigMap an:

```
[student@workstation ~]$ oc describe cm/appconfig  
Name:  appconfig  
...output omitted...  
  
Data  
====  
APP_MSG:  
---  
Elvis lives  
Events:  <none>
```

- 8.3. Führen Sie den Befehl `oc set env` aus, um der Deploymentkonfiguration die ConfigMap hinzuzufügen:

```
[student@workstation ~]$ oc set env deployment/elvis --from cm/appconfig  
deployment.apps/elvis updated
```

9. Überprüfen Sie, ob eine neue Bereitstellung ausgelöst wird, und warten Sie, bis der neue Anwendungs-Pod bereit ist und ausgeführt wird. Verifizieren Sie, ob der Schlüssel APP_MSG als eine Umgebungsvariable in den Container injiziert wird.

- 9.1. Verifizieren Sie, ob die neue Bereitstellung ausgelöst wurde:

```
[student@workstation ~]$ oc status
...output omitted...
deployment/elvis deploys istag/elvis:latest
  deployment #3 running for 21 seconds - 1 pod
  deployment #2 deployed 16 minutes ago
  deployment #1 deployed 16 minutes ago
...output omitted...
```

- 9.2. Warten Sie, bis der Anwendungs-Pod erneut bereitgestellt wird. Überprüfen Sie, ob der neue Anwendungs-Pod bereit ist und ausgeführt wird:

```
[student@workstation DO288-apps]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
elvis-66c7f6d47f-ll2jq   1/1     Running   0          2m36s
```

- 9.3. Verifizieren Sie, ob der Schlüssel APP_MSG als eine Umgebungsvariable in den Container injiziert wird:

```
[student@workstation ~]$ oc rsh elvis-66c7f6d47f-ll2jq env | grep APP_MSG
APP_MSG=Elvis lives
```

10. Testen Sie die Anwendung, indem Sie deren REST-API-URL (<http://elvis-youruser-design-container.apps.cluster.domain.example.com/api/hello>) aufrufen, und vergewissern Sie sich, dass der Schlüsselwert APP_MSG in der Antwort erscheint.

```
[student@workstation ~]$ curl \
  http://elvis-${RHT_OCP4_DEV_USER}-design-container.${RHT_OCP4_WILDCARD_DOMAIN}\ \
/api/hello
Hello world from host [elvis-66c7f6d47f-ll2jq]. Message received = Elvis lives
```

Bewertung

Verwenden Sie als Benutzer student auf dem Rechner workstation den Befehl lab, um Ihre Arbeit zu bewerten. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie den Befehl so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab design-container grade
```

Beenden

Führen Sie auf der workstation den Befehl lab design-container finish aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken. Die Aktion „Beenden“ veröffentlicht dieses Projekt und seine Ressourcen.

```
[student@workstation ~]$ lab design-container finish
```

Hiermit ist die praktische Übung abgeschlossen.

Zusammenfassung

In diesem Kapitel wurden die folgenden Themen behandelt:

- Zu den Deploymentoptionen für RHOC-P-Container gehören:
 - Vorab erstellte Container-Images direkt in einem OpenShift-Cluster bereitstellen
 - Ein Containerfile mit einem Basis-Image erstellen und an Ihre Anforderungen anpassen
 - Einen S2I-Build (Source-to-Image) verwenden, bei dem RHOC-P Quellcode mit einem Builder-Image kombiniert
- Häufige Änderungen an Containerfiles, die zum Ausführen eines Containers auf RHOC-P erforderlich sind:
 - Root-Gruppenberechtigungen für Dateien, die von Prozessen im Container gelesen oder geschrieben werden
 - Ausführbare Dateien müssen Ausführungsberechtigungen für die Gruppe aufweisen
 - Prozesse, die im Container ausgeführt werden, dürfen keine privilegierten Ports (Ports unter 1024) überwachen
- Secrets verwenden, um sensible Informationen und den Zugriff von Ihren Pods aus zu speichern
- ConfigMap-Ressourcen verwenden, um nicht sensible umgebungsspezifische Daten zu speichern

Kapitel 3

Veröffentlichen von Enterprise Container-Images

Ziel

Interagieren mit einer Unternehmens-Registry und Veröffentlichen von Container-Images in der Registry

Ziele

- Verwalten von Container-Images in Registries mit Linux-Container-Tools
- Zugreifen auf die interne OpenShift-Registry mit Linux-Container-Tools
- Erstellen von Image-Streams für Container-Images in externen Registries

Abschnitte

- Verwalten von Images in einer Unternehmens-Registry (und angeleitete Übung)
- Zulassen des Zugriffs auf die OpenShift-Registry (und angeleitete Übung)
- Erstellen von Image-Streams (und angeleitete Übung)

Praktische Übung

Veröffentlichen von Enterprise Container-Images

Verwalten von Images in einer Unternehmens-Registry

Ziele

Nach Abschluss dieses Abschnitts sollten Sie in der Lage sein, Container-Images in Registries mit Linux-Container-Tools zu verwalten.

Überprüfen von Container-Registries

Eine *Container-Image-Registry*, *Container-Registry* oder ein Registry-Server speichert die Images, die Sie als Container bereitstellen, und bietet Mechanismen zum Abrufen, Übertragen, Aktualisieren, Suchen und Entfernen von Container-Images. Dabei wird eine Standard-REST-API verwendet, die von der *Open Container Initiative (OCI)* definiert wird und auf der Docker Registry HTTP API v2 basiert. Aus der Perspektive einer Organisation, die einen OpenShift-Cluster ausführt, gibt es viele Arten von Container-Registries:

Öffentliche Registries

Registries, mit denen jeder Container-Images direkt aus dem Internet ohne Authentifizierung nutzen kann. Docker Hub, Quay.io und die Red Hat Registry sind Beispiele für öffentliche Container-Registries.

Private Registries

Registries, die nur ausgewählten Nutzern zur Verfügung stehen und in der Regel eine Authentifizierung erfordern. Die auf Bedingungen basierende Red Hat-Registry ist ein Beispiel für eine private Container-Registry.

Externe Registries

Registries, die Ihr Unternehmen nicht kontrolliert. Sie werden in der Regel von einem Cloud-Provider oder einem Software-Provider verwaltet. Quay.io ist ein Beispiel für eine externe Container-Registry.

Unternehmens-Registries

Registry-Server, die von Ihrer Organisation verwaltet werden. Sie stehen in der Regel nur den Mitarbeitern und Auftragnehmern der Organisation zur Verfügung.

Interne OpenShift-Registries

Ein von einem OpenShift-Cluster intern verwalteter Registry-Server zum Speichern von Container-Images. Erstellen Sie diese Images mit den OpenShift-Build-Konfigurationen und dem S2I-Prozess oder einem Containerfile, oder importieren Sie sie von anderen Registries.

Diese Arten von Registries schließen sich nicht gegenseitig aus: Eine Registry kann gleichzeitig sowohl eine öffentliche als auch eine private Registry sein. In der Regel ist eine öffentliche Registry auch eine externe Registry, da Ihr Unternehmen ohne Authentifizierung über das Internet darauf zugreifen kann und Ihre Organisation sie nicht kontrolliert.

Dieselbe Registry kann auch eine private Registry sein, wenn Ihr Unternehmen über einen Plan mit dem Registry-Anbieter verfügen, der Ihnen gestattet, private Images zu hosten, und Ihre Organisation auch die Kontrolle darüber hat, wer auf diese privaten Images zugreifen kann.

Quay.io fungiert für einige Benutzer sowohl als öffentliche als auch als private Registry. Derselbe Entwickler kann öffentliche Container-Images aus Quay.io und auch Container-Images von einem Anbieter verwenden, die eine Authentifizierung erfordern.

Von Red Hat verwaltete Registries

Red Hat verwaltet eine Reihe öffentlicher und privater Container-Registries, um verschiedene Arten von Container-Images für unterschiedliche Zielgruppen bereitzustellen. Auf Container-Images, die von Red Hat oder seinen Partnern mit SLAs auf Produktionsebene unterstützt werden, kann über den *Red Hat Container Catalog* zugegriffen werden. Auf Community- und nicht unterstützte Container-Images kann über Quay.io zugegriffen werden.

Der Red Hat Container Catalog unter <https://access.redhat.com/containers> ist eine Webbenutzeroberfläche, mit der Sie diese Registries anzeigen und durchsuchen können, um detaillierte Informationen über auf Red Hat Enterprise -Linux basierende Images zu erhalten.

Die von Red Hat bereitgestellten Images profitieren von der langen Erfahrung von Red Hat in Bezug auf die Verwaltung von Sicherheitsschwachstellen und Defekten in Red Hat Enterprise Linux und anderen Produkten. Das Red Hat-Sicherheitsteam schützt und kontrolliert diese qualitativ hochwertigen Images und signiert dann diese Images, um Manipulationen zu verhindern. Red Hat erstellt diese Images jedes Mal neu, wenn neue Schwachstellen entdeckt werden, und führt einen Qualitätssicherungsprozess durch.

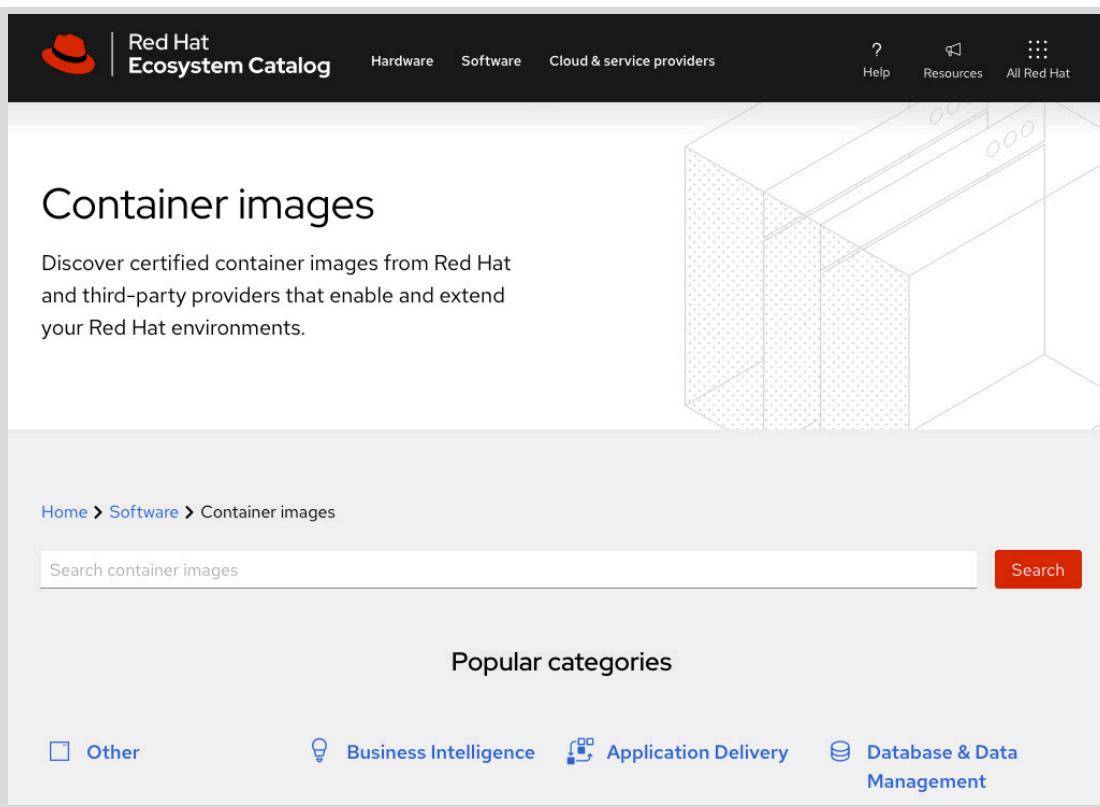


Abbildung 3.1: Red Hat Container Catalog

Geschäftskritische Anwendungen sollten so weit wie möglich auf diese vertrauenswürdigen und unterstützten Images zurückgreifen, anstatt Images aus anderen öffentlichen Registries zu verwenden. Diese anderen Images werden möglicherweise nicht angemessen getestet, gewartet oder aktualisiert, wenn neue Sicherheitsprobleme erkannt werden.

Der Red Hat Container Catalog bietet eine einheitliche Ansicht auf drei zugrunde liegende Container-Registries:

Red Hat Container Registry unter registry.access.redhat.com

Dies ist eine öffentliche Registry, die Images für Red Hat-Produkte hostet und keine Authentifizierung erfordert. Beachten Sie, dass diese Container-Registry zwar öffentlich ist, die meisten Container-Images, die Red Hat bereitstellt, erfordern jedoch, dass der Benutzer über eine aktive Red Hat-Produktsubskription verfügt und er die Endbenutzervereinbarung (EUA) des Produkts einhält. Nur eine Teilmenge der Images, die in der öffentlichen Registry von Red Hat verfügbar sind, können frei weitergegeben werden. Dies sind Images, die auf den Red Hat Enterprise Linux Universal Base Images (UBI) basieren.

Die auf Bedingungen basierende Red Hat Registry unter registry.redhat.io

Dies ist eine private Registry, die Images für Red Hat-Produkte hostet und eine Authentifizierung erfordert. Um Images daraus abzurufen, müssen Sie sich mit Ihren Anmelddaten beim Red Hat Customer Portal authentifizieren. Für gemeinsam genutzte Umgebungen wie OpenShift oder CI/CD Pipelines können Sie ein Servicekonto oder ein Authentifizierungstoken erstellen, um zu vermeiden, dass Ihre persönlichen Anmelddaten angezeigt werden.

Red Hat-Partner-Registry unter registry.connect.redhat.com

Dies ist eine private Registry, die Images für Drittanbieterprodukte von zertifizierten Partnern hostet. Zur Authentifizierung benötigt sie zudem Ihre Red Hat Customer Portal-Anmelddaten. Sie können nach dem Ermessen des Partners abonniert oder lizenziert werden.

Die Quay.io-Registry

Red Hat verwaltet auch die Quay.io-Container-Registry, bei der sich jeder für ein kostenloses Benutzerkonto registrieren und eigene Container-Images veröffentlichen kann.

Red Hat bietet keine Gewähr für Container-Images, die in Quay.io gehostet werden. Sie können von völlig ungepflegten, einmaligen Experimenten, über gute, stabile und ordnungsgemäß gepflegte Container-Images von Open-Source-Communities ohne Service Level Agreement (SLA) bis hin zu vollständig unterstützten Produkten von Anbietern, die kostenlosen, nicht authentifizierten Zugriff auf ihre Container-Images für Produkttests anbieten, reichen.

Die meisten Benutzer verwenden Quay.io als öffentliche Registry, doch Unternehmen können auch Tarife buchen, die die Verwendung von Quay.io als private Registry ermöglichen.

Bereitstellen von Enterprise Container-Registries

Der Zugriff auf externe, öffentliche oder private Registries über das Internet ist sehr praktisch, aber viele Organisationen gestatten Entwicklern nicht, externe Container-Images abzurufen und auszuführen. Diese Organisationen beschränken Entwickler auf eine Reihe von Container-Images, die Sicherheits-, Qualitäts- und Konformitätskriterien erfüllen.

Sich auf externe Registries zum Abrufen und Übertragen von Images für Ihre Produktionshosts zu verlassen ist nicht frei von Risiken. Wenn die Registry beispielsweise aufgrund eines Fehlers oder einer geplanten Wartung durch den Anbieter inaktiv ist, können Sie keine neuen Container bereitstellen. Im Falle eines Fehlers schlägt die automatische Skalierung von OpenShift ebenfalls fehl, da OpenShift nicht die Images abrufen kann, die zum Starten zusätzlicher Pods erforderlich sind. Abhängig von Ihrer Bandbreite kann das Übertragen von Images in das Internet oder das Abrufen aus dem Internet zudem sehr langsam erfolgen.

In einigen Organisationen sind Container-Images nur für die interne Verwendung vorgesehen und dürfen nicht veröffentlicht werden. Durch das Einrichten einer Unternehmens-Registry lösen Sie dieses Problem. Konfigurieren Sie nach dem Einrichten einer Unternehmens-Registry Containerhosts innerhalb der Organisation so, dass nur Images aus dieser Registry und nicht aus den standardmäßigen externen Registries abgerufen werden.

Kapitel 3 | Veröffentlichen von Enterprise Container-Images

Durch Ausführen eines Registry-Servers in Ihrer Organisation können Sie Risiken mindern und zusätzliche Funktionen implementieren. Sie können beispielsweise verschiedene Umgebungen erstellen und steuern, wer Images auf diese übertragen oder daraus abrufen kann. Sie können einen Genehmigungs-Workflow definieren, um validierte Images aus der Entwicklung in die Produktion zu verschieben. Sie können das Scannen von Schwachstellen implementieren und eine Benachrichtigung senden, wenn der Scanner einen Fehler in einem Image in der Produktion feststellt.

Registry-Server-Software

Zu den verfügbaren Registry-Servern zählen *Red Hat Quay Enterprise*, der Open-Source-Docker-Distribution-Server und Produkte wie JFrog und Nexus. OpenShift kann Container von jedem dieser Registry-Server bereitstellen.

Red Hat Quay Enterprise ist eine Container-Image-Registry mit erweiterten Funktionen, beispielsweise Image-Sicherheitsscan, rollenbasierter Zugriff, Organisations- und Teamverwaltung, Image-Build-Automatisierung, Auditing, Georeplikation und Hochverfügbarkeit.

Red Hat Quay Enterprise bietet eine Webbenutzeroberfläche und eine REST-API. Es kann als Container lokal, bei ausgewählten Cloud-Providern und auch auf *Red Hat OpenShift Container Platform* bereitgestellt werden. Es ist auch die Server-Software, auf der *Quay.io* basiert.

Wenn Sie *Quay Enterprise* auf *OpenShift* bereitstellen, ersetzt es nicht die interne Registry des Clusters. Eine *Quay Enterprise*-Instanz, die auf einem *OpenShift*-Cluster ausgeführt wird, ist für alle praktischen Zwecke wie jede andere *OpenShift*-Anwendung verfügbar und in der Regel für andere *OpenShift*-Cluster und alle anderen Containerhosts in Ihrer Organisation verfügbar.

The screenshot shows the Quay.io website interface. At the top, there is a navigation bar with links for RED HAT Quay.io, EXPLORE, APPLICATIONS, REPOSITORIES, and TUTORIAL. There is also a search bar and a user profile icon labeled 'yourquay...'. Below the navigation, there is a main search bar. Underneath it, a section titled 'APPLICATIONS AND REPOSITORIES' displays a list of four repositories:

Rank	Repository	Stars	Activity
1.	bitnami/sealed-secrets-controller	1	activity
2.	calico/node	22	activity
3.	jetstack/cert-manager-controller	11	activity
4.	jetstack/cert-manager-webhook	0	activity

Each repository entry includes a brief description, the last modified date, and a small icon representing its status or activity level.

Abbildung 3.2: Quay.io-Webseite

Zugreifen auf Container-Registries

Für den Zugriff auf eine öffentliche Registry von *OpenShift* aus ist in der Regel keine Konfiguration erforderlich, da eine öffentliche Registry ein TLS-Zertifikat enthalten sollte, das von einer vertrauenswürdigen Zertifizierungsstelle (Certificate Authority, CA) bereitgestellt wird.

Kapitel 3 | Veröffentlichen von Enterprise Container-Images

Für den Zugriff auf eine private Registry von OpenShift aus ist in der Regel eine zusätzliche Konfiguration zum Verwalten von Authentifizierungsanmeldedaten und -token erforderlich. Eine Unternehmens-Registry kann auch die Konfiguration interner CAs erfordern.

Verwenden Sie zum Zugreifen auf Container-Registries die *OCI Distribution API*, die auf der älteren Docker Registry API basiert. Red Hat empfiehlt zur Verwendung der API die Red Hat Enterprise Linux (RHEL)-Containertools zu verwenden: Podman, Buildah und Skopeo. Mit den RHEL-Containertools können Container-Images basierend auf OCI-Standards angepasst, bereitgestellt und debuggt werden.

Die OCI-Organisation (Open Container Initiative) legt offene Standards für das Container-Laufzeit- und Container-Image-Format sowie für zugehörige REST-APIs fest. Das OCI-Container-Image-Format ist ein Dateisystemordner mit diskreten Dateien, in denen das Manifest, die Metadaten und Layer von Container-Images gespeichert werden.

Verwalten von Containern mit Podman

Podman ist ein Tool für die Verwaltung von Pods und Containern, ohne dass ein Container-Daemon erforderlich ist, wodurch die Angriffsfläche reduziert und die Leistung verbessert wird. Pod- und Containerprozesse werden als untergeordnete Prozesse des Befehls `podman` erstellt.

Podman kann angehaltene Container neu starten sowie Container-Images übertragen, übergeben, konfigurieren, erzeugen und erstellen. Der Befehl `podman` folgt in der Regel der `docker`-Befehlssyntax und bietet zusätzliche Funktionen, z. B. die Verwaltung von Pods. Podman unterstützt keine `docker`-Befehle, die sich nicht auf die Container-Engine beziehen, wie z. B. den Swarm-Modus.

Authentifizieren mit Registries

Um auf eine private Registry zuzugreifen, müssen Sie sich in der Regel authentifizieren. Podman stellt den Sub-Befehl `login` bereit, der ein Zugriffstoken generiert und es für die nachfolgende Wiederverwendung speichert.

```
[user@host ~]$ podman login quay.io
Username: developer1
Password: MyS3cret!
Login Succeeded!
```

Nach erfolgreicher Authentifizierung speichert Podman ein Zugriffstoken in der Datei `/run/user/UID/containers/auth.json`. Das Pfadpräfix `/run/user/UID` ist nicht fest und stammt aus der Umgebungsvariable `XDG_RUNTIME_DIR`.

Sie können sich mit Podman gleichzeitig bei mehreren Registries anmelden. Jede neue Anmeldung fügt entweder ein Zugriffstoken in derselben Datei hinzu oder aktualisiert es. Jedes Zugriffstoken wird vom Registry-Server-FQDN indiziert.

Führen Sie den Sub-Befehl `logout` aus, um sich bei einer Registry abzumelden:

```
[user@host ~]$ podman logout quay.io
Remove login credentials for registry.redhat.io
```

Verwenden Sie die Option `--all`, um sich von allen Registries abzumelden und alle Zugriffstoken zu verwerfen, die zur Wiederverwendung gespeichert wurden:

```
[user@host ~]$ podman logout --all  
Remove login credentials for all registries
```

Skopeo und Buildah können ebenfalls die von Podman gespeicherten Authentifizierungstoken verwenden, aber sie können keine interaktive Passwortauflösung ausgeben.

Podman erfordert standardmäßig TLS und die Verifizierung des Remote-Zertifikats. Wenn Ihr Registry-Server entweder nicht für die Verwendung von TLS konfiguriert ist oder für die Verwendung eines selbstsignierten TLS-Zertifikats oder eines TLS-Zertifikats konfiguriert ist, das von einer unbekannten Zertifizierungsstelle signiert wurde, können Sie den Sub-Befehlen `login` und `pull` die Option `--tls-verify=false` hinzufügen.

Verwalten von Container-Registries mit Skopeo

Red Hat unterstützt den Befehl `skopeo` zum Verwalten von Images in einer Container-Image-Registry. Skopeo verwendet keine Container-Engine, daher ist es effizienter als die Sub-Befehle `tag`, `pull` und `push` von Podman.

Skopeo bietet auch zusätzliche Funktionen, die in Podman nicht enthalten sind, wie z. B. das Signieren und Löschen von Container-Images auf einem Registry-Server.

Der Befehl `skopeo` übernimmt einen Sub-Befehl, Optionen und Argumente:

```
[user@host ~]$ skopeo subcommand [options] location...
```

Wichtige Sub-Befehle

- `copy`, um Images von einem Speicherort an einen anderen zu kopieren.
- `delete`, um Images aus einer Registry zu löschen
- `inspect`, um Metadaten zu einem Image anzuzeigen

Wichtige Optionen

`--creds username:password`
Um der Registry Anmelddaten oder ein Authentifizierungstoken bereitzustellen

`--[src-|dest-]tls-verify=false`
Deaktiviert die TLS-Zertifikatsüberprüfung

Für die Authentifizierung bei privaten Registries kann Skopeo auch die Datei `auth.json` verwenden, die mit dem Befehl `podman login` erstellt wurde. Alternativ können Sie Ihre Anmelddaten in der Befehlszeile übergeben, wie unten gezeigt.

```
[user@host ~]$ skopeo inspect --creds developer1:MyS3cret! \  
docker://registry.redhat.io/rhscl/postgresql-96-rhel7
```



Warnung

Sie können Anmelddaten zwar für Befehlszeilertools bereitstellen, aber dadurch werden ein Eintrag in Ihrem Befehlsverlauf erstellt und weitere Sicherheitsprobleme hinzugefügt. Verwenden Sie geeignete Techniken, um das Übergeben von Anmelddaten als Text an Befehle zu vermeiden:

```
[user@host ~]$ read -p "PASSWORD: " -s password  
PASSWORD:  
[user@host ~]$ skopeo inspect --creds developer1:$password \  
docker://registry.redhat.io/rhscl/postgresql-96-rhel7
```

Skopeo verwendet URLs, um Speicherorte von Container-Images darzustellen, und URI-Schemas, um Container-Image-Formate und Registry-APIs darzustellen. Die folgende Liste zeigt die gängigsten URI-Schemas:

oci

Bezeichnet Container-Images, die in einem lokalen, in OCI formatierten Ordner gespeichert sind.

docker

Bezeichnet Remote-Container-Images, die auf einem Registry-Server gespeichert sind.

containers-storage

Bezeichnet Container-Images, die im lokalen Container-Engine-Cache gespeichert sind.

Übertragen und Kennzeichnen von Images auf einem Registry-Server

Mit dem Sub-Befehl `copy` von Skopeo können Container-Images direkt zwischen Registries kopiert werden, ohne die Image-Layer im lokalen Container-Storage zu speichern. Mit dem Befehl können auch Container-Images von der lokalen Container-Engine auf einen Registry-Server kopiert und diese Images in einem einzigen Vorgang mit Tags versehen werden.

So kopieren Sie ein Container-Image mit dem Namen `myimage` aus der lokalen Container-Engine in eine unsichere, öffentliche Registry in `registry.example.com` unter der Organisation oder dem Benutzerkonto `myorg`:

```
[user@host ~]$ skopeo copy --dest-tls-verify=false \  
containers-storage:myimage \  
docker://registry.example.com/myorg/myimage
```

So kopieren Sie ein Container-Image aus dem in OCI formatierten Ordner `/home/user/myimage` in eine unsichere, öffentliche Registry in `registry.example.com` unter der Organisation oder dem Benutzerkonto `myorg`:

```
[user@host ~]$ skopeo copy --dest-tls-verify=false \  
oci:/home/user/myimage \  
docker://registry.example.com/myorg/myimage
```

Beim Kopieren von Container-Images zwischen privaten Registries können Sie sich entweder mit Podman bei beiden Registries authentifizieren, bevor Sie den Sub-Befehl `copy` aufrufen, oder mit

Kapitel 3 | Veröffentlichen von Enterprise Container-Images

den Optionen `--src-creds` und `--dest-creds` die Authentifizierungsanmeldedaten angeben, wie unten gezeigt:

```
[user@host ~]$ skopeo copy --src-creds=testuser:testpassword \
--dest-creds=testuser1:testpassword \
docker://srcregistry.domain.com/org1/private \
docker://dstegistry.domain2.com/org2/private
```

Bei Argumenten für den Befehl `skopeo` handelt es sich immer um vollständige Image-Namen. Das folgende Beispiel enthält einen ungültigen Befehl, da er nur den Namen des Registry-Servers als Zielargument bereitstellt:

```
[user@host ~]$ skopeo copy oci:myimage \
docker://registry.example.com/
```

Mit dem Skopeo-Befehl `copy` können auch Images in Remote-Repositories mit Tags versehen werden. Im folgenden Beispiel wird ein vorhandenes Image mit dem Tag `1.0` als `latest` markiert:

```
[user@host ~]$ skopeo copy docker://registry.example.com/myorg/myimage:1.0 \
docker://registry.example.com/myorg/myimage:latest
```

Aus Effizienzgründen liest oder sendet Skopeo keine Image-Layer , die bereits am Zielort vorhanden sind. Es liest zuerst das Manifest des Quell-Images, bestimmt dann, welche Layer bereits am Ziel vorhanden sind, und kopiert dann nur die fehlenden Layer . Wenn Sie mehrere Images kopieren, die aus demselben übergeordneten Element erstellt wurden, kopiert Skopeo die übergeordneten Layer nicht mehrmals.

Löschen von Images aus einer Registry

Um das Container-Image `myorg/myimage` aus der Registry in `registry.example.com` zu löschen, führen Sie den folgenden Befehl aus:

```
[user@host ~]$ skopeo delete docker://registry.example.com/myorg/myimage
```

Der Sub-Befehl `delete` kann optional die Optionen `--creds` und `--tls-verify=false` übernehmen.

Authentifizieren von OpenShift bei privaten Registries

OpenShift erfordert auch Anmeldedaten für den Zugriff auf Container-Images in privaten Registries. Diese Anmeldedaten werden als Secrets gespeichert.

Sie können Ihre Anmeldedaten für die private Registry direkt an den Befehl `oc create secret` übergeben:

```
[user@host ~]$ oc create secret docker-registry registrycreds \
--docker-server registry.example.com \
--docker-username youruser \
--docker-password yourpassword
```

Eine andere Möglichkeit zum Erstellen des Secrets besteht darin, das Authentifizierungstoken aus dem Befehl `podman login` zu verwenden:

```
[user@host ~]$ oc create secret generic registrycreds \
--from-file dockerconfigjson=${XDG_RUNTIME_DIR}/containers/auth.json \
--type kubernetes.io/dockerconfigjson
```

Anschließend verknüpfen Sie das Secret mit dem Servicekonto **default** aus Ihrem Projekt:

```
[user@host ~]$ oc secrets link default registrycreds --for pull
```

Um das Secret für den Zugriff auf ein S2I-Builder-Image zu verwenden, verknüpfen Sie das Secret mit dem Servicekonto **builder** aus Ihrem Projekt:

```
[user@host ~]$ oc secrets link builder registrycreds
```



Literaturhinweise

Open Container Initiative (OCI)

<https://www.opencontainers.org/>

A Practical Introduction to Container Terminology

<https://developers.redhat.com/blog/2018/02/22/container-terminology-practical-introduction>

Red Hat Container-Registry-Authentifizierung

<https://access.redhat.com/RegistryAuthentication>

Weitere Informationen zu den RHEL-Containertools finden Sie im Handbuch *Building, running, and managing containers* für Red Hat Enterprise Linux 8 unter https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/building_running_and_managing_containers/index

► Angeleitete Übung

Verwenden einer Unternehmens-Registry

In dieser Übung interagieren Sie mit einem Registry-Server für Container-Images.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Übertragen von Images an eine externe, authentifizierte Container-Registry
- Bereitstellen einer containerisierten Anwendung in OpenShift mithilfe einer externen, authentifizierten Container-Registry als Eingabe

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen aktiven OpenShift-Cluster
- Die Befehle `podman` und `skopeo`.
- Auf OCI-konforme Dateien für das Beispiel-Container-Image `ubi-sleep`

Führen Sie den folgenden Befehl auf der `workstation`-VM aus, um die Voraussetzungen zu überprüfen und die Lösungsdateien herunterzuladen:

```
[student@workstation ~]$ lab external-registry start
```

Anweisungen

- 1. Melden Sie sich bei einer externen Registry an und übertragen Sie ein Image aus einem OCI-konformen Ordner auf der Festplatte.

- 1.1. Überprüfen Sie die Layer des `ubi-sleep`-Container-OCI-Images auf der lokalen Festplatte. OCI-Images werden als Dateisystemordner gespeichert, der mehrere Dateien enthält:

```
[student@workstation ~]$ ls ~/D0288/labs/external-registry/ubi-sleep
blobs index.json oci-layout
```

- 1.2. Laden Sie die Konfiguration Ihrer Kursumgebung.

Führen Sie den folgenden Befehl aus, um die in der ersten angeleiteten Übung erstellten Umgebungsvariablen zu laden:

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

- 1.3. Melden Sie sich mit Podman bei Ihrem persönlichen Quay.io-Benutzerkonto an. Podman fordert Sie auf, Ihr Quay.io-Passwort einzugeben.

```
[student@workstation ~]$ podman login -u ${RHT_OCP4_QUAY_USER} quay.io
Password:
Login Succeeded!
```

- 1.4. Kopieren Sie mit Skopeo das OCI-Image in die externe Registry auf Quay.io und kennzeichnen Sie es als 1.0.

Sie können auch den folgenden skopeo copy-Befehl aus dem Skript push-image.sh im Ordner /home/student/D0288/Labs/external-registry ausführen oder ausschneiden und einfügen. Veröffentlichen Sie dieses Repository nicht. Dieses Repository muss für die Übung privat bleiben.

```
[student@workstation ~]$ skopeo copy \
oci:/home/student/D0288/labs/external-registry/ubi-sleep \
docker://quay.io/${RHT_OCP4_QUAY_USER}/ubi-sleep:1.0
...output omitted...
Writing manifest to image destination
Storing signatures
```

- 1.5. Stellen Sie mit Podman sicher, dass das Image in der externen Registry vorhanden ist.



Anmerkung

Wenn Sie das Image in den podman search-Ergebnissen nicht finden können, fahren Sie mit dem nächsten Schritt fort. Quay.io kann Suchergebnisse kürzen, die zu viele Übereinstimmungen zurückgeben würden.

```
[student@workstation ~]$ podman search quay.io/ubi-sleep
INDEX      NAME
...output omitted...
quay.io    quay.io/yourquayuser/ubi-sleep
...output omitted...
```

- 1.6. Überprüfen Sie mit Skopeo das Image in der externen Registry:

```
[student@workstation ~]$ skopeo inspect \
docker://quay.io/${RHT_OCP4_QUAY_USER}/ubi-sleep:1.0
{
  "Name": "quay.io/yourquayuser/ubi-sleep",
  "Tag": "1.0",
  ...output omitted...
```

- 2. Vergewissern Sie sich, dass Podman Images aus der externen Registry ausführen kann.

- 2.1. Starten Sie einen Testcontainer aus dem Image in der externen Registry:

```
[student@workstation ~]$ podman run -d --name sleep \
quay.io/${RHT_OCP4_QUAY_USER}/ubi-sleep:1.0
Trying to pull quay.io/youruser/ubi-sleep:1.0...Getting image source signatures
...output omitted...
```

- 2.2. Vergewissern Sie sich, ob der neue Container ausgeführt wird:

```
[student@workstation ~]$ podman ps
CONTAINER ID        IMAGE               ...   NAMES
63c5167376e5      quay.io/youruser/ubi-sleep:1.0 ... sleep
```

- 2.3. Überprüfen Sie, ob der neue Container eine Logausgabe generiert:

```
[student@workstation ~]$ podman logs sleep
...output omitted...
sleeping
sleeping
```

- 2.4. Halten Sie den Testcontainer an und entfernen Sie ihn.

```
[student@workstation ~]$ podman stop sleep
...output omitted...
[student@workstation ~]$ podman rm sleep
...output omitted...
```

- 3. Stellen Sie anhand des Images aus der externen Registry eine Anwendung in OpenShift bereit:

- 3.1. Melden Sie sich bei OpenShift an und erstellen Sie ein neues Projekt. Stellen Sie dem Projektnamen Ihren Entwicklerbenutzernamen voran.

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-external-registry
Now using project "youruser-docker-build" on server "https://
api.cluster.domain.example.com:6443".
```

- 3.2. Versuchen Sie, eine Anwendung aus dem Container-Image in der externen Registry bereitzustellen. Dies schlägt fehl, da OpenShift Anmeldedaten benötigt, um auf die externe Registry zuzugreifen.

```
[student@workstation ~]$ oc new-app --name sleep \
--docker-image quay.io/${RHT_OCP4_QUAY_USER}/ubi-sleep:1.0
error: unable to locate any local docker images with name "quay.io/yourquayuser/
ubi-sleep:1.0"
...output omitted...
```

- 3.3. Erstellen Sie ein Secret aus dem Zugriffstoken der Container-Registry-API, das von Podman gespeichert wurde.

Sie können auch den folgenden `oc create secret`-Befehl aus dem Skript `create-secret.sh` ausführen oder ausschneiden und in das Verzeichnis `/home/student/D0288/labs/external-registry` einfügen.

Kapitel 3 | Veröffentlichen von Enterprise Container-Images

```
[student@workstation ~]$ oc create secret generic quayio \
--from-file .dockerconfigjson=${XDG_RUNTIME_DIR}/containers/auth.json \
--type kubernetes.io/dockerconfigjson
secret/quayio created
```

3.4. Verknüpfen Sie das neue Secret mit dem Servicekonto default.

```
[student@workstation ~]$ oc secrets link default quayio --for pull
```

3.5. Stellen Sie eine Anwendung aus dem Container-Image in der externen Registry bereit. Diesmal kann OpenShift auf die externe Registry zugreifen.

```
[student@workstation ~]$ oc new-app --name sleep \
--docker-image quay.io/${RHT_OCP4_QUAY_USER}/ubi-sleep:1.0
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "sleep" created
deployment.apps "sleep" created
--> Success
...output omitted...
```

3.6. Warten Sie, bis der Anwendungs-Pod bereit ist und ausgeführt wird:

```
[student@workstation ~]$ oc get pod
NAME           READY   STATUS    RESTARTS   AGE
sleep-7bf77b7596-ldrsvr  1/1     Running   0          95s
```

3.7. Überprüfen Sie, ob der Pod ein AusgabeLog generiert:

```
[student@workstation ~]$ oc logs sleep-7bf77b7596-ldrsvr
...output omitted...
sleeping
sleeping
```

- ▶ 4. Löschen Sie das Projekt in OpenShift sowie den Container und das Image in der externen Container-Registry. Da Quay.io die Wiederherstellung alter Container-Images ermöglicht, müssen Sie Ihr Repository ebenfalls auf Quay.io löschen.

4.1. Löschen Sie das OpenShift-Projekt:

```
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-external-registry
project.project.openshift.io "external-registry" deleted
```

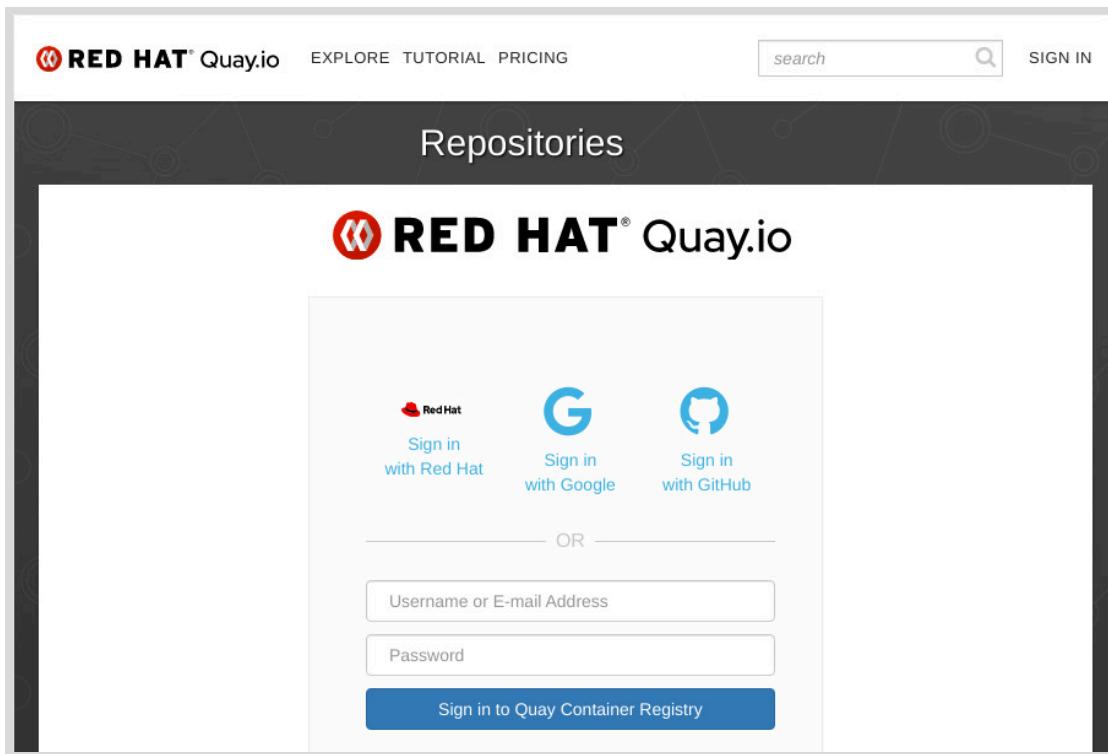
4.2. Löschen Sie das Container-Image aus der externen Registry:

```
[student@workstation ~]$ skopeo delete \
docker://quay.io/${RHT_OCP4_QUAY_USER}/ubi-sleep:1.0
```

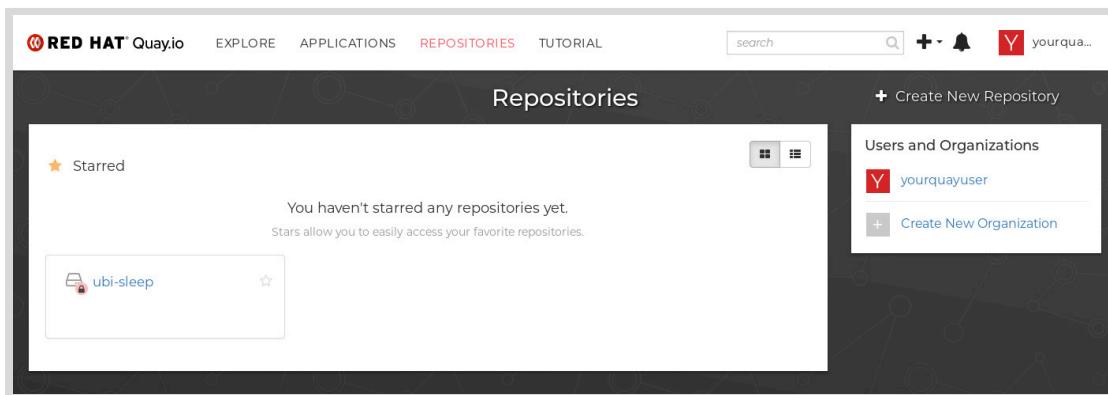
4.3. Melden Sie sich bei Quay.io mit Ihrem persönlichen kostenlosen Benutzerkonto an.

Kapitel 3 | Veröffentlichen von Enterprise Container-Images

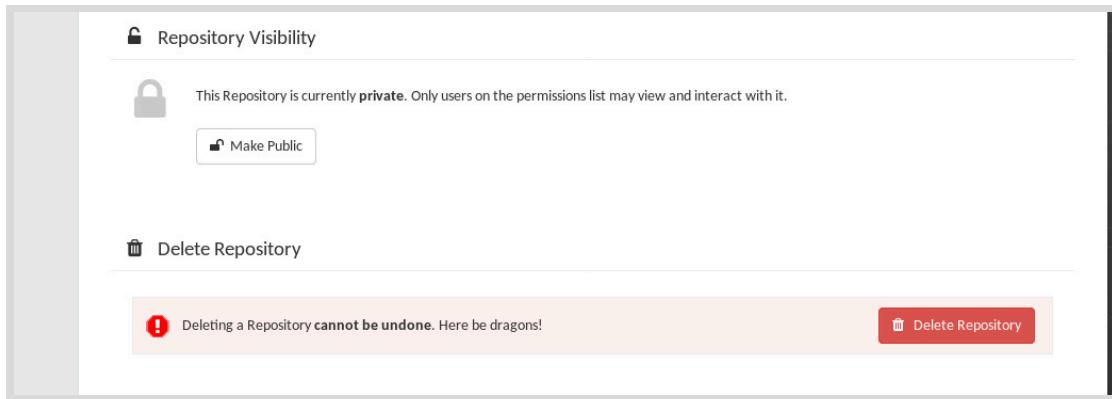
Navigieren Sie zu <http://quay.io> und klicken Sie auf **Sign In**, um Ihre Benutzeranmelddaten einzugeben. Klicken Sie auf **Sign in to Quay Container Registry**, um sich bei Quay.io anzumelden.



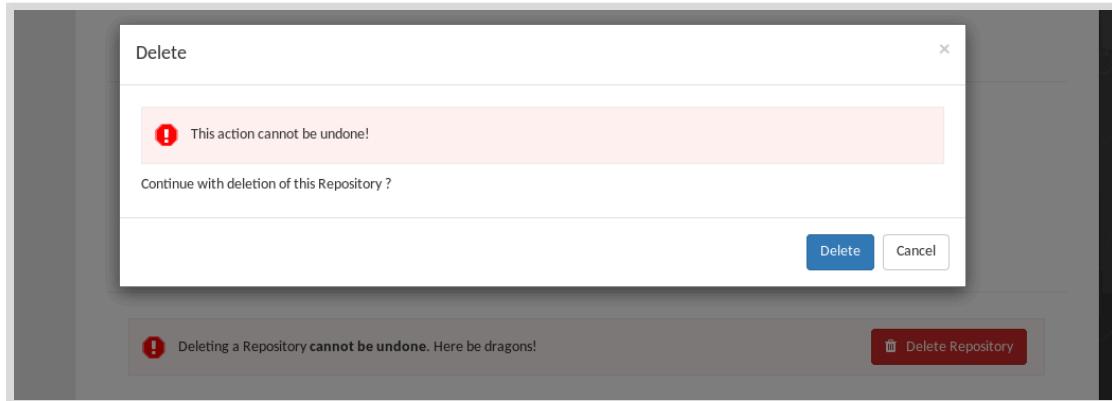
- 4.4. Klicken Sie im Hauptmenü von Quay.io auf **Repositories** und suchen Sie nach **ubi-sleep**. Das Schlosssymbol zeigt an, dass es sich um ein privates Repository handelt, das eine Authentifizierung für Abrufe und Übertragungen erfordert. Klicken Sie auf **ubi-sleep**, um die Seite **Repository Activity** anzuzeigen.



- 4.5. Scrollen Sie auf der Seite **Repository Activity** für das **ubi-sleep**-Repository nach unten und klicken Sie auf das Zahnradsymbol, um den den **Settings**-Tab anzuzeigen. Scrollen Sie nach unten und klicken Sie auf **Delete Repository**.



- 4.6. Klicken Sie im Dialogfeld **Delete** auf **Delete**, um das Löschen des `ubi-sleep`-Repository zu bestätigen. Nach einigen Augenblicken werden Sie zur Seite **Repositories** zurückgeleitet. Sie können sich jetzt bei Quay.io abmelden.



Beenden

Führen Sie auf der `workstation`-VM den Befehl `lab external-registry finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken. Die Aktion „Beenden“ veröffentlicht dieses Projekt und seine Ressourcen.

```
[student@workstation ~]$ lab external-registry finish
```

Hiermit ist die angeleitete Übung beendet.

Zulassen des Zugriffs auf die OpenShift-Registry

Ziele

Nach Abschluss dieses Abschnitts sollten Sie in der Lage sein, auf die interne OpenShift-Registry mit Linux-Container-Tools zuzugreifen.

Überprüfen der internen Registry

OpenShift führt einen internen Registry-Server aus, um Entwickler-Workflows basierend auf Source-to-Image (S2I) zu unterstützen. Entwickler erstellen neue Container-Images mit S2I, indem sie eine Build-Konfiguration mit dem Befehl `oc new-app` und anderen Mitteln erstellen. Eine Build-Konfiguration erstellt Container-Images entweder aus Quellcode oder einem Containerfile und speichert sie in der internen Registry.

Entwickler müssen die interne Registry nicht verwenden. Sie könnten ihre Container-Images lokal erstellen, indem sie die Red Hat Enterprise Linux-Container-Tools verwenden und sie in eine externe öffentliche oder private Registry übertragen, auf die OpenShift zugreifen kann. Sie können auch ihre Build-Konfigurationen so einrichten, dass das endgültige Image direkt an eine externe öffentliche oder private Registry übertragen wird.

Das OpenShift-Installationsprogramm stellt standardmäßig eine interne Registry bereit, sodass Entwickler mit der Entwicklungsarbeit beginnen können, sobald ihnen der OpenShift-Cluster zur Verfügung steht. Ohne eine interne Registry müssten Entwickler warten, bis ein Registry-Server bereitgestellt wird und ihnen die Anmelddaten für den Zugriff zur Verfügung gestellt werden. Sie müssten zudem lernen, wie Secrets für den Zugriff auf private Registries konfiguriert werden, bevor sie Entwicklungsarbeiten durchführen können.

Für den Zugriff auf eine interne OpenShift-Registry außerhalb des S2I-Prozesses sind einige Anwendungsfälle vorhanden. Beispiel:

- Eine Organisation erstellt Container-Images bereits lokal und ist noch nicht bereit, ihren Entwicklungs-Workflow zu ändern. Diese Organisationen verfügen möglicherweise über eine private Registry mit eingeschränkten Funktionen und möchten diese durch die interne OpenShift-Registry ersetzen.
- Eine Organisation unterhält mehrere OpenShift-Cluster und muss Container-Images aus einem Entwicklungs- in einen Produktionscluster kopieren. Diese Organisationen verfügen möglicherweise über ein CI/CD-Tool, das Images aus einer internen Registry entweder in eine externe Registry oder eine andere interne Registry weiterleitet.
- Ein unabhängiger Softwareanbieter (ISV) erstellt Container-Images für seine Kunden und veröffentlicht sie in einer privaten von einem Cloud-Serviceanbieter wie Quay.io verwalteten Registry oder in einer vom ISV verwalteten Unternehmens-Registry.

Die interne OpenShift-Registry kann alle Funktionen bereitstellen, die ein Kunde benötigt, z. B. detaillierte Zugriffssteuerungen basierend auf OpenShift-Benutzern, -Gruppen und -Rollen. Die interne Registry bietet möglicherweise bessere Funktionen oder eine größere Benutzerfreundlichkeit im Vergleich zur aktuellen Unternehmens-Registry einer Organisation, z. B. wenn sie auf dem Docker-Distribution-Registry-Server basiert. Diese Organisationen könnten ihre aktuelle Registry auslaufen lassen und die interne OpenShift-Registry als neue Unternehmens-Registry verwenden.

Kapitel 3 | Veröffentlichen von Enterprise Container-Images

Andere Kunden benötigen möglicherweise erweiterte Funktionen, wie Image-Sicherheitsscan und Georeplikation, und einen leistungsfähigeren Unternehmens-Registry-Server wie Red Hat Quay Enterprise. Kunden, die eine leistungsfähigere Unternehmens-Registry einführen, müssen möglicherweise die interne Registry weiterhin verfügbar halten, um Images in die Unternehmens-Registry der Organisation kopieren zu können.

Der Image-Registry-Operator

Das OpenShift-Installationsprogramm konfiguriert die interne Registry so, dass nur innerhalb des OpenShift-Clusters darauf zugegriffen werden kann. Das Bereitstellen der internen Registry für den externen Zugriff ist eine einfache Prozedur, erfordert jedoch Administrationsberechtigungen für den Cluster.

Der OpenShift *Image Registry Operator* verwaltet die interne Registry. Alle Konfigurationseinstellungen für den Image-Registry-Operator befinden sich in der Konfigurationsressource `cluster` im Projekt `openshift-image-registry`. Ändern Sie das Attribut `spec.defaultRoute` in `true` und der Image-Registry-Operator erstellt eine Route, um die interne Registry verfügbar zu machen. Eine Möglichkeit, diese Änderung durchzuführen, verwendet den folgenden `oc patch`-Befehl:

```
[user@host ~] oc patch config.imageregistry cluster -n openshift-image-registry \
--type merge -p '{"spec":{"defaultRoute":true}}'
```

Die Route `default-route` verwendet den standardmäßigen Platzhalter-Domain-Name für Anwendungen, die im Cluster bereitgestellt werden:

```
[user@host ~] oc get route -n openshift-image-registry
NAME           HOST/PORT
default-route   default-route-openshift-image-registry.domain.example.com ...
```

Authentifizieren bei einer internen Registry

Um sich mit den Linux-Container-Tools bei einer internen Registry anzumelden, müssen Sie das OpenShift-Authentifizierungstoken Ihres Benutzers abrufen.

Verwenden Sie den Befehl `oc whoami -t`, um das Token abzurufen. Das Token besteht aus einer langen, zufälligen Zeichenfolge. Wenn Sie das Token als Shell-Variable speichern, ist es einfacher, Befehle einzugeben:

```
[user@host ~] TOKEN=$(oc whoami -t)
```

Verwenden Sie das Token als Teil eines `login`-Sub-Befehls von Podman:

```
[user@host ~] podman login -u myuser -p ${TOKEN} \
default-route-openshift-image-registry.domain.example.com
```

Sie können das Token auch als Wert der `--[src|dst]creds`-Optionen von Skopeo verwenden.

```
[user@host ~] skopeo inspect --creds=myuser:${TOKEN} \
docker://default-route-openshift-image-registry.domain.example.com/...
```

Zugreifen auf die interne Registry als sichere oder unsichere Registry

Wenn Ihr OpenShift-Cluster mit einem gültigen TLS-Zertifikat für seine Platzhalter-Domain konfiguriert ist, können Sie die Linux-Container-Tools verwenden, um mit Images in jedem Projekt zu arbeiten, auf das Sie Zugriff haben.

Im folgenden Beispiel wird Skopeo verwendet, um das Container-Image für die Anwendung `myapp` im Projekt `myproj` zu überprüfen. Es wird davon ausgegangen, dass ein vorheriger `podman login`-Befehl erfolgreich war.

```
[user@host ~] skopeo inspect \
docker://default-route-openshift-image-registry.domain.example.com/myproj/myapp
```

Wenn Ihr OpenShift-Cluster die Zertifizierungsstelle (Certification Authority, CA) verwendet, die das OpenShift-Installationsprogramm standardmäßig generiert, müssen Sie auf die interne Registry als unsichere Registry zugreifen:

```
[user@host ~] skopeo inspect --tls-verify=false \
docker://default-route-openshift-image-registry.domain.example.com/myproj/myapp
```

Ein Cluster-Administrator kann die Route für die interne Registry auf unterschiedliche Weise konfigurieren, z. B. mithilfe einer internen Zertifizierungsstelle (CA), die von Ihrer Organisation verwaltet wird. In diesem Szenario kann Ihre Entwickler-Workstation bereits so konfiguriert sein, dass sie dem TLS-Zertifikat der internen Registry vertraut.

Ihr Unternehmen ruft möglicherweise auch das öffentliche Zertifikat der internen Zertifizierungsstelle des OpenShift-Clusters ab und deklariert es in Ihrem Unternehmen als vertrauenswürdig. Der Cluster-Administrator kann eine alternative Route mit einem kürzeren Hostnamen einrichten, um die interne Registry verfügbar zu machen.

Diese Szenarien werden im Rahmen dieses Kurses nicht behandelt. Weitere Informationen zur Konfiguration von TLS-Zertifikaten für OpenShift und Ihre lokale Container-Engine finden Sie in *Red Hat Training-Kursen zur OpenShift-Administration*, z. B. *Red Hat OpenShift Administration I* (DO280) und *Red Hat Security: Securing Containers and OpenShift* (DO425).

Gewähren des Zugriffs auf Images in einer internen Registry

Jeder Benutzer mit Zugriff auf ein OpenShift-Projekt kann Images an dieses Projekt übertragen und aus diesem Projekt abrufen, je nach Zugriffsebene. Wenn Benutzer im Projekt über die Rolle `admin` oder `edit` verfügen, können sie Images sowohl aus diesem Projekt abrufen als auch an dieses Projekt übertragen. Wenn sie stattdessen nur über die Rolle `view` für das Projekt verfügen, können sie Images nur aus diesem Projekt abrufen.

OpenShift bietet auch einige spezielle Rollen für den Fall, dass Sie nur Zugriff auf Images innerhalb eines Projekts gewähren möchten und keinen Zugriff zum Ausführen anderer Entwicklungsaufgaben, wie z. B. das Erstellen und Bereitstellen von Anwendungen innerhalb des Projekts, erteilen möchten. Die häufigsten dieser Rollen sind:

`registry-viewer` und `system:image-puller`

Mit diesen Rollen können Benutzer Images aus der internen Registry abrufen und untersuchen.

registry-editor und system:image-pusher

Mit diesen Rollen können Benutzer Images an die interne Registry übertragen und kennzeichnen.

Die **system:***-Rollen bieten die minimalen Funktionen, die zum Abrufen und Übertragen von Images an die interne Registry erforderlich sind. Wie bereits erwähnt, benötigen OpenShift-Benutzer, die bereits über die Rollen **admin** oder **edit** in einem Projekt verfügen, diese **system:***-Rollen nicht.

Die **registry-***-Rollen bieten umfassendere Funktionen rund um die Registry-Verwaltung für Organisationen, die die interne Registry als Unternehmens-Registry verwenden möchten. Diese Rollen gewähren zusätzliche Rechte, z. B. das Erstellen neuer Projekte, jedoch keine anderen Rechte, beispielsweise das Erstellen und Bereitstellen von Anwendungen. Die OCI-Standards geben nicht an, wie eine Image-Registry verwaltet werden soll, daher muss jeder, der eine interne OpenShift-Registry verwaltet, über OpenShift-Verwaltungskonzepte und den Befehl `oc` Bescheid wissen. Dadurch werden die **registry-***-Rolle weniger nützlich.

Im folgenden Beispiel kann ein Benutzer Images aus der internen Registry in einem bestimmten Projekt abrufen. Sie benötigen entweder projekt- oder clusterweiten Administratorzugriff, um den Befehl `oc policy` verwenden zu können.

```
[user@host ~] oc policy add-role-to-user system:image-puller \
user_name -n project_name
```

Allgemeine OpenShift-Authentifizierungs- und Autorisierungskonzepte werden im Rahmen dieses Kurses nicht behandelt. Weitere Informationen zur Konfiguration von TLS-Zertifikaten für OpenShift und Ihre lokale Container-Engine finden Sie in *Red Hat Training-Kursen zur OpenShift-Administration*, z. B. *Red Hat OpenShift Administration I* (DO280) und *Red Hat Security: Securing Containers and OpenShift* (DO425).

**Literaturhinweise**

Weitere Informationen zu Verfügbarkeitseinstellungen der internen Registry finden Sie im Kapitel *Image Registry Operator in OpenShift Container Platform* des Handbuchs *Registry* für Red Hat OpenShift Container Platform 4.6 unter https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/registry/index#configuring-registry-operator

Weitere Informationen über das Gewähren des Zugriffs auf Images in der internen Registry finden Sie im Kapitel *Accessing the registry* des Handbuchs *Registry* für Red Hat OpenShift Container Platform 4.6 unter https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/registry/index#accessing-the-registry

► Angeleitete Übung

Zulassen des Zugriffs auf die OpenShift-Registry

In dieser Übung greifen Sie auf die interne OpenShift-Registry mit Linux-Container-Tools zu.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Übertragen eines Images an eine interne Registry mit Linux-Container-Tools
- Erstellen eines Containers aus einer internen Registry mit Linux-Container-Tools

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen ausgeführten OpenShift-Cluster mit bereitgestellter interner Registry
- Die Befehle `podman` und `skopeo`.
- Auf OCI-konforme Dateien für das Beispiel-Container-Image `ubi - info`

Führen Sie den folgenden Befehl auf der `workstation`-VM aus, um die Voraussetzungen zu überprüfen und die Lösungsdateien herunterzuladen:

```
[student@workstation ~]$ lab expose-registry start
```

Anweisungen

- 1. Stellen Sie sicher, dass die interne Registry Ihres OpenShift-Clusters verfügbar ist.

- 1.1. Laden Sie die Konfiguration Ihrer Kursumgebung.

Führen Sie den folgenden Befehl aus, um die in der ersten angeleiteten Übung erstellten Umgebungsvariablen zu laden:

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

- 1.2. Melden Sie sich bei OpenShift mit Ihrem Entwicklerbenutzerkonto an.

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

- 1.3. Überprüfen Sie, ob im Projekt `openshift-image-registry` eine Route vorhanden ist. Die Ausgabe des Befehls `oc route` wurde bearbeitet, um aus Gründen der

Kapitel 3 | Veröffentlichen von Enterprise Container-Images

Lesbarkeit jede Spalte in einer Zeile anzuzeigen, da der Hostname zu lang ist, um in die Papierbreite zu passen.

```
[student@workstation ~]$ oc get route -n openshift-image-registry
NAME          HOST/PORT
...output omitted...
default-route  default-route-openshift-image-
registry.apps.cluster.domain.example.com
...output omitted...
```

**Anmerkung**

Eine standardmäßige Red Hat OpenShift Container Platform-Installation erlaubt es einem regulären Benutzer nicht, Ressourcen in den `openshift-*`-Projekten anzuzeigen. Der Cluster dieses Kursraums weist allen Entwicklerbenutzerkonten der Kursteilnehmer zusätzliche Rechte zu, damit sie den vorherigen Vorgang ausführen können.

- Um die Eingabe zu vereinfachen, speichern Sie den Hostnamen der Route der internen Registry in einer Shell-Variable.

Sie können den Befehl `oc get` aus dem Skript `push-image.sh` im Ordner `/home/student/D0288/labs/expose-registry` ausschneiden und einfügen.

```
[student@workstation ~]$ INTERNAL_REGISTRY=$( oc get route default-route \
-n openshift-image-registry -o jsonpath='{.spec.host}' )
```

- Überprüfen Sie den Wert Ihrer Shell-Variable `INTERNAL_REGISTRY`. Er sollte mit der Ausgabe des ersten `oc get route`-Befehls übereinstimmen.

```
[student@workstation ~]$ echo ${INTERNAL_REGISTRY}
default-route-openshift-image-registry.apps.cluster.domain.example.com
```

- 2. Übertragen Sie mit Ihrem Entwicklerbenutzerkonto ein Image an die interne OpenShift-Registry.

- Erstellen Sie ein Projekt zum Hosten der Image-Streams, die die Container-Images verwalten, die Sie an die interne OpenShift-Registry übertragen:

```
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-common
Now using project "youruser-common" on server
"https://api.cluster.domain.example.com:6443".
```

- Rufen Sie das OpenShift-Authentifizierungstoken Ihres Entwicklerbenutzerkontos für die spätere Verwendung mit Befehlen ab:

```
[student@workstation ~]$ TOKEN=$(oc whoami -t)
```

- Verifizieren Sie, ob der Ordner `ubi-info` ein in OCI formatiertes Container-Image enthält:

```
[student@workstation ~]$ ls ~/D0288/labs/expose-registry/ubi-info
blobs  index.json  oci-layout
```

- 2.4. Kopieren Sie mit Skopeo das OCI-Image in die interne Registry des Kursraumclusters und kennzeichnen Sie es als 1.0. Verwenden Sie den Hostnamen und das Token, die in den vorherigen Schritten abgerufen wurden.

Sie können den Befehl `skopeo copy` aus dem Skript `push-image.sh` im Ordner `/home/student/D0288/labs/expose-registry` ausschneiden und einfügen.

```
[student@workstation ~]$ skopeo copy \
--dest-creds=${RHT_OCP4_DEV_USER}:${TOKEN} \
oci:/home/student/D0288/labs/expose-registry/ubi-info \
docker://${INTERNAL_REGISTRY}/${RHT_OCP4_DEV_USER}-common/ubi-info:1.0
...output omitted...
Writing manifest to image destination
Storing signatures
```

- 2.5. Überprüfen Sie, ob zur Verwaltung des neuen Container-Images ein Image-Stream erstellt wurde. Die Ausgabe des Befehls `oc get is` wurde bearbeitet, um aus Gründen der Lesbarkeit jede Spalte in einer Zeile anzuzeigen, da der Name des Image-Repository zu lang ist, um in die Papierbreite zu passen.

```
[student@workstation ~]$ oc get is
NAME      IMAGE REPOSITORY
ubi-info  default-route-openshift-image-registry.apps...
...output omitted...
```

- 3. Erstellen Sie einen lokalen Container anhand des Images in der internen OpenShift-Registry.
- 3.1. Melden Sie sich bei der internen OpenShift-Registry mit dem Authentifizierungstoken von *Schritt 2.2* und dem Registry-Hostnamen von *Schritt 1.4* an:

```
[student@workstation ~]$ podman login -u ${RHT_OCP4_DEV_USER} \
-p ${TOKEN} ${INTERNAL_REGISTRY}
Login Succeeded!
```

- 3.2. Laden Sie das Container-Image `ubi-info:1.0` in die lokale Container-Engine herunter.

```
[student@workstation ~]$ podman pull \
${INTERNAL_REGISTRY}/${RHT_OCP4_DEV_USER}-common/ubi-info:1.0
...output omitted...
Writing manifest to image destination
Storing signatures
...output omitted...
```

- 3.3. Starten Sie einen neuen Container aus dem Container-Image `ubi-info:1.0`. Der Container zeigt Systeminformationen wie den Hostnamen und den freien

Kapitel 3 | Veröffentlichen von Enterprise Container-Images

Speicherplatz an und wird dann beendet. Für den aktiven Container spezifische Informationen werden in der folgenden Ausgabe ausgelassen:

```
[student@workstation ~]$ podman run --name info \
${INTERNAL_REGISTRY}/${RHT_OCP4_DEV_USER}-common/ubi-info:1.0
...output omitted...
--- Host name:
...output omitted...
--- Free memory
...output omitted...
--- Mounted file systems (partial)
...output omitted...
```

- 4. Führen Sie eine Bereinigung durch. Löschen Sie alle während dieser Übung erstellten Ressourcen.

- 4.1. Löschen Sie das Container-Image aus der internen Registry Ihres Kursraumclusters, indem Sie den Image-Stream löschen:

```
[student@workstation ~]$ oc delete is ubi-info
imagestream.image.openshift.io "ubi-info" deleted
```

- 4.2. Löschen Sie das OpenShift-Projekt:

```
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-common
project.project.openshift.io "youruser-common" deleted
```

- 4.3. Löschen Sie den Testcontainer und das Image aus der lokalen Container-Engine:

```
[student@workstation ~]$ podman rm info
...output omitted...
[student@workstation ~]$ podman rmi -f \
${INTERNAL_REGISTRY}/${RHT_OCP4_DEV_USER}-common/ubi-info:1.0
...output omitted...
```

Beenden

Führen Sie auf der workstation-VM den Befehl `lab expose-registry finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken. Die Aktion „Beenden“ veröffentlicht dieses Projekt und seine Ressourcen.

```
[student@workstation ~]$ lab expose-registry finish
```

Hiermit ist die angeleitete Übung beendet.

Erstellen von Image-Streams

Ziele

Nach Abschluss dieses Abschnitts sollten Sie in der Lage sein, Image-Streams für Container-Images in externen Registries zu erstellen.

Beschreiben von Image-Streams

Image-Streams sind eines der Hauptunterscheidungsmerkmale zwischen OpenShift und Upstream-Kubernetes. Kubernetes-Ressourcen verweisen direkt auf Container-Images, OpenShift-Ressourcen, z. B. Deploymentkonfigurationen und Build-Konfigurationen, verweisen auf Image-Streams. OpenShift erweitert Kubernetes-Ressourcen, wie StatefulSet- und CronJob-Ressourcen, um Annotationen, die ihnen dabei helfen, mit OpenShift-Image-Streams zu arbeiten. Es ist zwar auch möglich, Image Streams mit Kubernetes-Bereitstellungen zu verwenden, dies ist jedoch in der Regel nicht erforderlich und wird in diesem Kurs nicht verwendet.

Image-Streams ermöglichen OpenShift, reproduzierbare, stabile Bereitstellungen von containerisierten Anwendungen und ebenso Rollbacks von Bereitstellungen auf den letzten als fehlerfrei bekannten Zustand sicherzustellen.

Image-Streams bieten einen stabilen, kurzen Namen zum Verweisen auf ein Container-Image, der unabhängig von Registry-Server- und Containerlaufzeitkonfigurationen ist.

Beispielsweise kann eine Organisation zunächst Container-Images direkt aus der öffentlichen Registry von Red Hat herunterladen und später eine Unternehmens-Registry als Spiegelung dieser Images einrichten, um Bandbreite zu sparen. OpenShift-Benutzer würden keine Änderung bemerken, da sie immer noch auf diese Images mit dem gleichen Image-Stream-Namen verweisen. Benutzer der RHEL-Containertools würden die Änderung bemerken, da sie entweder die Registry-Namen in ihren Befehlen oder ihre Container-Engine-Konfigurationen ändern müssten, um zuerst nach der lokalen Spiegelung zu suchen.

Es gibt andere Szenarien, in denen sich der von einem Image-Stream bereitgestellte Umweg als hilfreich erweist. Angenommen, Sie beginnen mit einem Datenbankcontainer-Image, das Sicherheitsprobleme aufweist, und der Anbieter benötigt zu lange, um das Image mit Korrekturen zu aktualisieren. Später finden Sie einen anderen Anbieter, der ein alternatives Container-Image für dieselbe Datenbank bereitstellt, bei dem diese Sicherheitsprobleme bereits behoben sind. Dieser Anbieter weist zudem eine bessere Erfolgsbilanz bei der Bereitstellung zeitnaher Updates auf. Wenn diese Container-Images in Bezug auf die Konfiguration von Umgebungsvariablen und Volumes kompatibel sind, können Sie einfach Ihren Image-Stream so ändern, dass er auf das Image des alternativen Anbieters zeigt.

Red Hat bietet erprobte, unterstützte Container-Images, die hauptsächlich als sofort einsatzbereiter Ersatz von Container-Images aus einigen beliebten Open-Source-Projekten wie der MariaDB-Datenbank funktionieren.

Beschreiben von Image-Stream-Tags

Ein Image-Stream stellt eine oder mehrere Gruppen von Container-Images dar. Jeder Satz, oder Stream, wird durch ein *Image-Stream-Tag* identifiziert. Im Gegensatz zu Container-Images auf einem Registry-Server, die mehrere Tags aus demselben Image-Repository (oder Benutzer oder

Kapitel 3 | Veröffentlichen von Enterprise Container-Images

Organisation) haben, kann ein Image-Stream mehrere Image-Stream-Tags aufweisen, die auf Container-Images aus verschiedenen Registry-Servern und Image-Repositories verweisen.

Ein Image-Stream stellt Standardkonfigurationen für eine Reihe von Image-Stream-Tags bereit. Jedes Image-Stream-Tag verweist auf einen Stream von Container-Images und kann die meisten Konfigurationen aus dem zugehörigen Image-Stream überschreiben.

Ein Image-Stream-Tag speichert eine Kopie der Metadaten über das aktuelle Container-Image und kann optional eine Kopie der aktuellen und vergangenen Container-Image-Layer speichern. Das Speichern von Metadaten ermöglicht eine schnellere Suche und Überprüfung von Container-Images, da Sie nicht den Quell-Registry-Server zurückgreifen müssen.

Durch das Speichern von Image-Layern kann ein Image-Stream-Tag als lokaler Image-Cache fungieren, sodass diese Layer nicht von ihrem Quell-Registry-Server abgerufen werden müssen. Das Image-Stream-Tag speichert seine zwischengespeicherten Image-Layer in der internen Registry von OpenShift. Nutzer des zwischengespeicherten Images, z. B. Pods und Deploymentkonfigurationen, verweisen einfach auf die interne Registry als Quell-Registry des Images.

Es gibt einige andere OpenShift-Ressourcentypen im Zusammenhang mit Image-Streams, aber Entwickler können sie in der Regel als Implementierungsdetails der internen Registry vernachlässigen und sich nur um Image-Streams und Image-Stream-Tags kümmern.

Um die Beziehung zwischen Image-Streams und Image-Stream-Tags besser zu visualisieren, können Sie das Projekt `openshift` untersuchen, das in allen OpenShift-Clustern vorab erstellt wird. Sie können dort sehen, dass es eine Reihe von Image-Streams in diesem Projekt gibt, einschließlich des Image-Streams `php`:

```
[user@host ~]$ oc get is -n openshift -o name
...output omitted...
imagestream.image.openshift.io/nodejs
imagestream.image.openshift.io/perl
imagestream.image.openshift.io/php
imagestream.image.openshift.io/postgresql
imagestream.image.openshift.io/python
...output omitted...
```

Für den Image-Stream `php` sind einige Tags und für jeden ist eine Image-Stream-Ressource vorhanden:

```
[user@host ~]$ oc get istag -n openshift | grep php
php:7.2      image-registry ...    6 days ago
php:7.3      image-registry ...    6 days ago
php:latest   image-registry ...    6 days ago
```

Wenn Sie den Befehl `oc describe` für einen Image-Stream ausführen, werden Informationen sowohl aus dem Image-Stream als auch aus den Image-Stream-Tags angezeigt:

```
[user@host ~]$ oc describe is php -n openshift
Name:                  php
Namespace:             openshift
...output omitted...
Tags:                  3
```

```
7.3 (latest)
tagged from registry.redhat.io/rhscl/php-73-rhel7:latest
...output omitted...
7.2
tagged from registry.redhat.io/rhscl/php-72-rhel7:latest
...output omitted...
```

Im vorherigen Beispiel bezieht sich jedes php-Image-Stream-Tag auf einen anderen Image-Namen.

Beschreiben von Image-Namen, -Tags und -IDs

Der Textname eines Container-Images ist einfach eine Zeichenfolge. Dieser Name wird manchmal so interpretiert, als ob er aus mehreren Komponenten besteht, z. B. `registry-host-name/repository-or-organization-or-user-name/image-name:tag-name`, aber das Aufteilen des Image-Namens in seine Komponenten ist nur eine Frage der Konvention, nicht der Struktur.

Eine Image-ID identifiziert ein unveränderliches Container-Image mit einem SHA-256-Hash eindeutig. Denken Sie daran, dass Sie ein Container-Image nicht ändern können. Stattdessen erstellen Sie ein neues Container-Image mit einer neuen ID. Wenn Sie ein neues Container-Image an einen Registry-Server übertragen, ordnet der Server den vorhandenen Textnamen der neuen Image-ID zu.

Wenn Sie einen Container über einen Image-Namen starten, laden Sie das Image herunter, das derzeit diesem Image-Namen zugeordnet ist. Die tatsächliche Image-ID hinter diesem Namen kann sich jederzeit ändern, und der nächste Container, den Sie starten, hat möglicherweise eine andere Image-ID. Wenn das Image, das einem Image-Namen zugeordnet ist, Probleme aufweist und Sie nur den Image-Namen kennen, können Sie kein Rollback auf ein früheres Image durchführen.

Image-Stream-Tags von OpenShift speichern den Verlauf der letzten Image-IDs, die sie von einem Registry-Server abgerufen haben. Der Verlauf von Image-IDs ist der Stream von Images aus einem Image-Stream-Tag. Sie können den Verlauf in einem Image-Stream-Tag verwenden, um zu einem vorherigen Image zurückzukehren, wenn z. B. ein neues Container-Image einen Deploymentfehler verursacht.

Durch das Aktualisieren eines Container-Images in einer externen Registry wird ein Image-Stream-Tag nicht automatisch aktualisiert. Das Image-Stream-Tag behält den Verweis auf die zuletzt abgerufene Image-ID bei. Dieses Verhalten ist für die Skalierung von Anwendungen von entscheidender Bedeutung, da es OpenShift von Änderungen isoliert, die auf einem Registry-Server stattfinden.

Angenommen, Sie stellen eine Anwendung aus einer externen Registry bereit und nach einigen Tagen des Testens mit einigen Benutzern entscheiden Sie sich, die Bereitstellung zu skalieren, um eine größere Benutzerpopulation zu ermöglichen. In der Zwischenzeit aktualisiert Ihr Anbieter das Container-Image in der externen Registry. Wenn OpenShift keine Image-Stream-Tags hätte, würden die neuen Pods das neue Container-Image erhalten, das sich vom Image auf dem ursprünglichen Pod unterscheidet. Abhängig von den Änderungen kann dies dazu führen, dass die Anwendung fehlschlägt. Da OpenShift die Image-ID des Original-Images in einem Image-Stream-Tag speichert, kann es neue Pods mit derselben Image-ID erstellen und jegliche Inkompatibilität zwischen dem ursprünglichen und dem aktualisierten Image vermeiden.

OpenShift behält die für den ersten Pod verwendete Image-ID bei und stellt sicher, dass neue Pods dieselbe Image-ID verwenden. OpenShift stellt sicher, dass alle Pods exakt dasselbe Image verwenden.

Kapitel 3 | Veröffentlichen von Enterprise Container-Images

Um die Beziehung zwischen einem Image-Stream, einem Image-Stream-Tag, einem Image-Namen und einer Image-ID besser zu visualisieren, sehen Sie sich den folgenden `oc describe is`-Befehl an, der das Quell-Image und die aktuelle Image-ID für jedes Image-Stream-Tag anzeigt:

```
[user@host ~]$ oc describe is php -n openshift
Name:          php
Namespace:     openshift
...output omitted...
7.3 (latest)
  tagged from registry.redhat.io/rhscl/php-73-rhel7:latest
  ...output omitted...
  * registry.redhat.io/rhscl/php-73-rhel7@sha256:22ba...09b5
  ...output omitted...
7.2
  tagged from registry.redhat.io/rhscl/php-72-rhel7:latest
  ...output omitted...
  * registry.redhat.io/rhscl/php-72-rhel7@sha256:e8d6...e615
  ...output omitted...
```

Wenn Ihr OpenShift-Cluster-Administrator das Image-Stream-Tag `php:7.3` bereits aktualisiert hat, zeigt der Befehl `oc describe is` mehrere Image-IDs für dieses Tag an:

```
[user@host ~]$ oc describe is php -n openshift
Name:          php
Namespace:     openshift
...output omitted...
7.3 (latest)
  tagged from registry.redhat.io/rhscl/php-73-rhel7:latest
  ...output omitted...
  * registry.redhat.io/rhscl/php-73-rhel7@sha256:22ba...09b5
  ...output omitted...
  registry.redhat.io/rhscl/php-73-rhel7@sha256:bc61...1e91
  ...output omitted...
7.2
  tagged from registry.redhat.io/rhscl/php-72-rhel7:latest
  ...output omitted...
  * registry.redhat.io/rhscl/php-72-rhel7@sha256:e8d6...e615
  ...output omitted...
```

Im vorherigen Beispiel gibt das Sternchen (*) an, welche Image-ID die aktuelle für jedes Image-Stream-Tag ist. Es ist in der Regel die letzte, die importiert werden soll, also die erste ID in der Liste.

Wenn ein Image-Stream-Tag von OpenShift auf ein Container-Image aus einer externen Registry verweist, müssen Sie das Image-Stream-Tag explizit aktualisieren, um neue Image-IDs aus der externen Registry zu erhalten. Standardmäßig überwacht OpenShift keine externen Registries auf Änderungen an der Image-ID, die einem Image-Namen zugeordnet ist.

Sie können ein Image-Stream-Tag konfigurieren, um die externe Registry nach einem definierten Zeitplan auf Aktualisierungen zu überprüfen. Standardmäßig werden neue Image-Stream-Tags nicht auf aktualisierte Images überprüft.

Kapitel 3 | Veröffentlichen von Enterprise Container-Images

Build-Konfigurationen aktualisieren automatisch das Image-Stream-Tag, das sie als Ausgabe-Image verwenden. Dies erzwingt die erneute Bereitstellung von Anwendungs-Pods mithilfe dieses Image-Stream-Tags.

Verwalten von Image-Streams und -Tags

Um eine Ressource für ein Image-Stream-Tag für ein Container-Image zu erstellen, das auf einer externen Registry gehostet ist, verwenden Sie den Befehl `oc import-image` mit den Optionen `--confirm` und `--from`. Der folgende Befehl aktualisiert ein Image-Stream-Tag oder erstellt eines, falls nicht vorhanden:

```
[user@host ~]$ oc import-image myimagestream[:tag] --confirm \
--from registry/myorg/myimage[:tag]
```

Wenn Sie keinen Tag-Namen angeben, wird standardmäßig das Tag `latest` verwendet. In diesem Beispiel verweist das Image-Stream-Tag auf den Image-Stream `myimagestream`. Wenn der entsprechende Image-Stream noch nicht vorhanden ist, erstellt OpenShift ihn.



Anmerkung

Um einen Image-Stream für Container-Images zu erstellen, die auf einem Registry-Server gehostet werden, der nicht mit einem vertrauenswürdigen TLS-Zertifikat eingerichtet sind, fügen Sie dem Befehl `oc import-image` die Option `--insecure` hinzu.

Der Tag-Name für das Image-Stream-Tag kann sich vom Container-Image-Tag auf dem Quell-Registry-Server unterscheiden. Im folgenden Beispiel wird ein 1.0-Image-Stream-Tag aus dem Tag `latest` des Quell-Registry-Servers erstellt (durch Weglassen):

```
[user@host ~]$ oc import-image myimagestream:1.0 --confirm \
--from registry/myorg/myimage
```

Um eine Image-Stream-Tag-Ressource für jedes Container-Image-Tag zu erstellen, das auf dem Quell-Registry-Server vorhanden ist, fügen Sie dem Befehl `oc import-image` die Option `--all` hinzu:

```
[user@host ~]$ oc import-image myimagestream --confirm --all \
--from registry/myorg/myimage
```

Sie können den Befehl `oc import-image` für einen vorhandenen Image-Stream ausführen, um eines seiner aktuellen Image-Stream-Tags auf die aktuellen Image-IDs auf dem Quell-Registry-Server zu aktualisieren.

```
[user@host ~]$ oc import-image myimagestream[:tag] --confirm
```

Durch das Aktualisieren eines Image-Streams mit der Option `--all` werden alle Image-Stream-Tags aktualisiert und zudem neue Image-Stream-Tags für neue Tags erstellt, die auf dem Quell-Registry-Server vorhanden sind.

Mit dem Befehl `oc tag` erhalten Sie eine bessere Kontrolle über ein Image-Stream-Tag. Er ermöglicht Änderungen wie:

Kapitel 3 | Veröffentlichen von Enterprise Container-Images

- Zuordnen eines Image-Stream-Tags zu einem anderen Registry-Server als demjenigen, der dem Image-Stream zugeordnet ist
- Zuordnen eines Image-Stream-Tags zu einem anderen Container-Image-Namen und -Tag
- Zuordnen eines Image-Stream-Tags zu einer bestimmten Image-ID, die möglicherweise nicht diejenige ist, die derzeit diesem Image-Tag auf dem Registry-Server zugeordnet ist
- Zuordnen eines Image-Streams zu einem anderen Image-Stream-Tag. Das vorherige Beispiel des Befehls `oc describe is` zeigt, dass das Image-Stream-Tag `php:latest` dem Image-Stream-Tag `php:7.3` folgt. Auf diese Weise können Aliase für Image-Stream-Tags erstellt werden.

Es liegt außerhalb des Rahmens dieses Kurses, alle Aspekte der Image-Stream-Verwaltung zu behandeln, obwohl einige von ihnen, z. B. Image-Änderungsereignisse, in späteren Kapiteln besprochen werden.

Verwenden von Image-Streams mit privaten Registries

Zum Erstellen von Image-Streams und Image-Stream-Tags, die auf eine private Registry verweisen, benötigt OpenShift ein Zugriffstoken für diesen Registry-Server.

Sie stellen dieses Zugriffstoken als Secret bereit, genauso wie Sie eine Anwendung aus einer privaten Registry bereitstellen würden, und Sie müssen das Secret nicht mit einem Servicekonto verknüpfen. Der Befehl `oc import-image` durchsucht die Secrets im aktuellen Projekt nach einem, das mit dem Registry-Hostnamen übereinstimmt.

Im folgenden Beispiel wird Podman verwendet, um sich bei einer privaten Registry anzumelden, ein Secret zum Speichern des Zugriffstokens sowie einen Image-Stream zu erstellen, der auf die private Registry verweist:

```
[user@host ~]$ podman login -u myuser registry.example.com
[user@host ~]$ oc create secret generic regtoken \
--from-file .dockerconfigjson=${XDG_RUNTIME_DIR}/containers/auth.json \
--type kubernetes.io/dockerconfigjson
[user@host ~]$ oc import-image myis --confirm \
--from registry.example.com/myorg/myimage
```

Nachdem Sie einen Image-Stream erstellt haben, können Sie ihn verwenden, um eine Anwendung mit dem Befehl `oc new-app -i myis` bereitzustellen. Sie können diesen Image-Stream auch als Builder-Image im Befehl `oc new-app myis~giturl` verwenden.

Standardmäßig ist eine Image-Stream-Ressource nur zum Erstellen von Anwendungen oder Builds im gleichen Projekt verfügbar.

Freigeben eines Image-Streams für mehrere Projekte

Es ist eine gängige Praxis, Projekte in OpenShift zu erstellen, um Ressourcen zu speichern, die von mehreren Benutzern und Entwicklungsteams gemeinsam genutzt werden. Diese gemeinsam genutzten Projekte speichern Ressourcen wie Image-Streams und Vorlagen, auf die sich Entwickler beim Bereitstellen von Anwendungen in ihren Projekten beziehen.

OpenShift wird mit einem gemeinsam genutzten Projekt namens `openshift` ausgeliefert, das Schnellstart-Anwendungsvorlagen sowie Image-Streams für S2I-BUILDER für gängige Programmiersprachen wie Python und Ruby bereitstellt. Einige Organisationen erstellen ähnliche Projekte für ihre Teams, anstatt dem `openshift`-Projekt Ressourcen hinzuzufügen, um Probleme beim Upgrade des Clusters auf eine neue Version von OpenShift zu vermeiden.



Wichtig

Sie hatten in früheren Versionen die Option, dem openshift-Projekt neue Ressourcen hinzuzufügen, aber ab Red Hat OpenShift Container Platform 4 wird das openshift-Projekt vom Samples-Operator verwaltet, der von Ihnen manuell hinzugefügte Ressourcen jederzeit entfernen kann.

Zum Erstellen und Bereitstellen von Anwendungen mithilfe eines Image-Streams, der in einem anderen Projekt definiert ist, haben Sie zwei Optionen:

- Erstellen Sie ein Secret mit einem Zugriffstoken für die private Registry in jedem Projekt, das den Image-Stream verwendet, und verknüpfen Sie dieses Secret mit den Servicekonten jedes Projekts.
- Erstellen Sie ein Secret mit einem Zugriffstoken für die private Registry nur für das Projekt, in dem Sie den Image-Stream erstellen, und konfigurieren Sie diesen Image-Stream mit einer lokalen Referenzrichtlinie. Gewähren Sie Rechte für die Verwendung des Image-Streams für Servicekonten aus jedem Projekt, das den Image-Stream verwendet.

Die erste Option ähnelt der Bereitstellung einer Anwendung aus einem Container-Image in einer privaten Registry. Dabei werden einige der Vorteile von Image-Streams nicht genutzt, denn wenn sich das Image-Stream-Tag ändert, auf das Sie sich beziehen, um auf einen anderen Registry-Server zu verweisen, müssen Sie in allen Projekten ein neues Secret für den neuen Registry-Server erstellen.

Mit der zweiten Option können Projekte, die auf den Image-Stream verweisen, von Änderungen in den von ihnen genutzten Image-Stream-Tags isoliert bleiben. Die zusätzliche Arbeit beim Zuweisen von Berechtigungen zu Servicekonten ergibt sich aus der Tatsache, dass Servicekonten über Rechte verfügen, die eingeschränkter sind als das Benutzerkonto, durch das sie erstellt werden.

Im folgenden Beispiel wird die zweite Option veranschaulicht. Sie erstellt einen Image-Stream im Projekt `shared` und verwendet diesen Image-Stream, um eine Anwendung im Projekt `myapp` bereitzustellen.

```
[user@host ~]$ podman login -u myuser registry.example.com
[user@host ~]$ oc project shared
[user@host ~]$ oc create secret generic regtoken \
--from-file .dockerconfigjson=${XDG_RUNTIME_DIR}/containers/auth.json \
--type kubernetes.io/dockerconfigjson
[user@host ~]$ oc import-image myis --confirm \
--reference-policy local \ ①
--from registry.example.com/myorg/myimage
[user@host ~]$ oc policy add-role-to-group system:image-puller \ ②
system:serviceaccounts:myapp ③
[user@host ~]$ oc project myapp
[user@host ~]$ oc new-app -i shared/myis
```

- ① Die Option `--reference-policy local` des Befehls `oc import-image`. Der Image-Stream wird so konfiguriert, dass Image-Layer in der internen Registry zwischengespeichert werden, sodass Projekte, die auf den Image-Stream verweisen, kein Zugriffstoken für die externe private Registry benötigen.

Kapitel 3 | Veröffentlichen von Enterprise Container-Images

- ❷ Die Rolle `system:image-puller` ermöglicht es einem Servicekonto, die Image-Layer abzurufen, die der Image-Stream in der internen Registry zwischengespeichert hat.
- ❸ Die Gruppe `system:serviceaccounts:myapp`. Diese Gruppe beinhaltet alle Servicekonten aus dem Projekt `myapp`. Der Befehl `oc policy` kann auf Benutzer und Gruppen verweisen, die noch nicht vorhanden sind.

Der Befehl `oc policy` erfordert keine Cluster-Administratorberechtigungen. Es sind nur Projektadministratorberechtigungen erforderlich.

Der Image-Stream aus dem vorherigen Beispiel kann auch das Builder-Image für den Befehl `oc new-app shared/myis~giturl` bereitstellen.



Literaturhinweise

Weitere Informationen über Image-Streams, Image-Stream-Tags und Image-IDs finden Sie im Kapitel *Understanding Containers, Images, and Imagestreams* des Handbuchs *Images* für Red Hat OpenShift Container Platform 4.6 unter https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/images/index#understanding-images

► Angeleitete Übung

Erstellen eines Image-Streams

In dieser Übung stellen Sie mithilfe eines Image-Streams eine auf Nginx basierende „hello, world“-Anwendung bereit.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Veröffentlichen eines Images aus einer externen Registry als ein Image-Stream in OpenShift
- Bereitstellen einer Anwendung mit dem Image-Stream

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen aktiven OpenShift-Cluster
- Auf das „hello, world“-Anwendungscontainer-Image (`redhattraining/hello-world-nginx`).

Führen Sie den folgenden Befehl auf der `workstation`-VM aus, um die Voraussetzungen zu überprüfen:

```
[student@workstation ~]$ lab image-stream start
```

Anweisungen

- 1. Melden Sie sich bei OpenShift an und erstellen Sie ein Projekt, um den Image-Stream für das Nginx-Container-Image zu hosten.
- 1.1. Laden Sie die Konfiguration Ihrer Kursumgebung.
Führen Sie den folgenden Befehl aus, um die in der ersten angeleiteten Übung erstellten Umgebungsvariablen zu laden:

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

- 1.2. Melden Sie sich bei OpenShift mit Ihrem Entwicklerbenutzerkonto an.

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

- 1.3. Erstellen Sie ein Projekt, um die Image-Streams zu hosten, die potenziell für mehrere Projekte freigegeben werden:

```
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-common
Now using project "youruser-common" on server
"https://api.cluster.domain.example.com:6443".
```

- 2. Erstellen Sie einen Image-Stream, der über die externe Registry auf das Nginx-Image verweist.
- 2.1. Stellen Sie sicher, dass das Image `redhattraining/hello-world-nginx` aus Quay.io ein einzelnes Tag mit dem Namen `latest` aufweist.

```
[student@workstation ~]$ skopeo inspect \
docker://quay.io/redhattraining/hello-world-nginx
{
    "Name": "quay.io/redhattraining/hello-world-nginx",
    "Tag": "latest",
    "Digest": "sha256:4f4f...acc1",
    "RepoTags": [
        "latest"
    ],
...output omitted...
```

- 2.2. Erstellen Sie den Image-Stream `hello-world`, der auf das Container-Image `redhattraining/hello-world-nginx` aus Quay.io verweist:

```
[student@workstation ~]$ oc import-image hello-world --confirm \
--from quay.io/redhattraining/hello-world-nginx
imagestream.image.openshift.io/hello-world imported
...output omitted...
Name:          hello-world
Namespace:     youruser-common
...output omitted...
Unique Images: 1
Tags:          1

latest
tagged from quay.io/redhattraining/hello-world-nginx
...output omitted...
```

- 2.3. Überprüfen Sie, ob das Image-Stream-Tag `hello-world:latest` erstellt wurde:

```
[student@workstation ~]$ oc get istag
NAME           IMAGE REF
... 
hello-world:latest  quay.io/redhattraining/hello-world-nginx@sha256:4f4f...acc1
```

- 2.4. Überprüfen Sie, ob der Image-Stream und sein Tag Metadaten über das Nginx-Container-Image enthalten:

```
[student@workstation ~]$ oc describe is hello-world
Name:          hello-world
Namespace:     youruser-common
```

Kapitel 3 | Veröffentlichen von Enterprise Container-Images

```
...output omitted...
Tags:          1

latest
tagged from quay.io/redhattraining/hello-world-nginx ①

* quay.io/redhattraining/hello-world-nginx@sha256:4f4f...acc1 ②
  2 minutes ago

...output omitted...
```

- ① Das Image-Stream-Tag `hello-world:latest` verweist auf ein Image aus Quay.io.
- ② Das Sternchen (*) gibt an, dass das `latest` Image-Stream-Tag nur auf das Image mit dem angegebenen SHA-256-Identifikator verweist.

► 3. Erstellen Sie ein neues Projekt und stellen Sie eine Anwendung mit dem Image-Stream `hello-world` aus dem Projekt `youruser-common` bereit.

3.1. Erstellen Sie ein Projekt zum Hosten der Testanwendung:

```
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-image-stream
Now using project "youruser-image-stream" on server
"https://api.cluster.domain.example.com:6443".
```

3.2. Stellen Sie eine Anwendung aus dem Image-Stream bereit:

```
[student@workstation ~]$ oc new-app --name hello \
-i ${RHT_OCP4_DEV_USER}-common/hello-world
--> Found image 44eaa13 (20 hours old) in image stream "youruser-common/hello-
world" under tag "latest" for "youruser-common/hello-world"
...output omitted...
--> Creating resources ...
  imagestreamtag.image.openshift.io "hello:latest" created
  deployment.apps "hello" created
  service "hello" created
--> Success
...output omitted...
```

3.3. Warten Sie, bis der Anwendungs-Pod bereit ist und ausgeführt wird:

```
[student@workstation ~]$ oc get pod
NAME           READY   STATUS    RESTARTS   AGE
hello-6599bb7b9c-zk58m  1/1     Running   0          40s
```

3.4. Erstellen Sie eine Route, um die Anwendung bereitzustellen:

```
[student@workstation ~]$ oc expose svc hello
route.route.openshift.io/hello exposed
```

3.5. Rufen Sie den Hostnamen der Route ab:

```
[student@workstation ~]$ oc get route  
NAME      HOST/PORT  
hello     hello-youruser-image-stream.apps.cluster.domain.example.com ...
```

- 3.6. Testen Sie die Anwendung mit dem Befehl `curl` und dem Hostnamen aus dem vorherigen Schritt.

```
[student@workstation ~]$ curl \  
http://hello-${RHT_OCP4_DEV_USER}-image-stream.${RHT_OCP4_WILDCARD_DOMAIN}  
...output omitted...  
<h1>Hello, world from nginx!</h1>  
...output omitted...
```

- 4. Löschen Sie die Projekte `youruser-common` und `youruser-image-stream`.

```
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-image-stream  
project.project.openshift.io "youruser-image-stream" deleted  
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-common  
project.project.openshift.io "youruser-image-common" deleted
```

Beenden

Führen Sie auf der `workstation`-VM den Befehl `lab image-stream finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken. Die Aktion „Beenden“ veröffentlicht dieses Projekt und seine Ressourcen.

```
[student@workstation ~]$ lab image-stream finish
```

Hiermit ist die angeleitete Übung beendet.

► Praktische Übung

Veröffentlichen von Enterprise Container-Images

In dieser praktischen Übung veröffentlichen Sie ein in OCI formatiertes Container-Image in einer externen Registry und stellen eine Anwendung aus diesem Image mit einem Image-Stream bereit.



Anmerkung

Für den am Ende jeder praktischen Übung verwendeten Befehl `grade` ist es erforderlich, dass Sie die exakten Projektnamen und anderen Identifikatoren, wie in der Spezifikation der praktischen Übung angegeben, verwenden.

Ergebnisse

Sie sollten in der Lage sein, eine Anwendung aus einem in einer externen Registry gespeicherten Image zu erstellen.

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen aktiven OpenShift-Cluster
- Auf OCI-konforme Dateien für das Beispiel-Container-Image (`php-info`)

Führen Sie den folgenden Befehl auf der `workstation`-VM aus, um die Voraussetzungen zu überprüfen und die Lösungsdateien herunterzuladen:

```
[student@workstation ~]$ lab expose-image start
```

Anforderungen

Das Anwendungs-Image umfasst eine PHP-Anwendung, die die Webserverumgebung und die PHP-Interpreter-Konfiguration anzeigt. Stellen Sie die Anwendung wie folgt bereit:

- Hosten Sie den Image-Stream für das außerhalb von OpenShift erstellt Image in einem Projekt mit dem Namen `youruser-common`.

Der Name des Container-Images lautet `php-info`. Die OCI-formatierten Image-Ebenen und das Manifest befinden sich im Ordner `/home/student/D0288/labs/expose-image/php-info`. Übertragen Sie dieses Image in ein privates Image-Repository in Ihrem Quay.io-Benutzerkonto.

- Stellen Sie die Anwendung in einem Projekt namens `youruser-expose-image` bereit.

Die Ressourcen der Anwendung werden als `info` bezeichnet. Greifen Sie auf die Anwendung mit dem standardmäßigen Hostnamen zu, der von OpenShift zugewiesen wurde.

Kapitel 3 | Veröffentlichen von Enterprise Container-Images

- Verwenden Sie die Konfigurationsdatei `/usr/local/etc/ocp4.config`, um Kursraumkonfigurationsdaten wie die Master-API-URL des OpenShift-Clusters abzurufen.

Anweisungen

1. Übertragen Sie das in OCI formatierte `php-info`-Container-Image an Quay.io.
2. Erstellen Sie als Entwicklerbenutzer das Projekt `youruser-common`, um den Image-Stream zu hosten, der auf das Image in der externen Registry zeigt. Erstellen Sie ein Secret mit Anmeldedaten für Quay.io und erstellen Sie den `php-info`-Image-Stream.
Erstellen Sie den Image-Stream mit der Option `--reference-policy local`, damit auch andere Projekte, die diesen Image-Stream verwenden, das im Projekt `youruser-common` gespeicherte Secret verwenden können.
3. Erstellen Sie ein neues Projekt mit dem Namen `youruser-expose-image`. Erstellen Sie anschließend eine neue Anwendung, indem Sie das Container-Image aus dem Image-Stream bereitstellen.
Gewähren Sie die Rolle `system:image-puller` im Projekt `youruser-common` allen Servicekonten im Projekt `youruser-expose-image`. Mit dieser Rolle können Pods aus einem Projekt Image-Streams aus einem anderen Projekt verwenden. Das Skript `grant-puller-role.sh` im Ordner `/home/student/D0288/labs/expose-image` enthält den Befehl `oc policy`, der diesen Vorgang ausführt.
4. Stellen Sie die Anwendung bereit und testen Sie sie. Überprüfen Sie, ob die Anwendung die PHP-Interpreter-Konfigurationsseite zurückgibt.
5. Bewerten Sie Ihre Arbeit.
Führen Sie den folgenden Befehl auf der `workstation`-VM aus, um zu überprüfen, ob alle Aufgaben abgeschlossen wurden:

```
[student@workstation ~]$ lab expose-image grade
```

6. Löschen Sie die OpenShift-Projekte und entfernen Sie das Image-Repository aus Quay.io.

Beenden

Führen Sie auf der `workstation`-VM den Befehl `lab expose-image finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken. Die Aktion „Beenden“ veröffentlicht dieses Projekt und seine Ressourcen.

```
[student@workstation ~]$ lab expose-image finish
```

Hiermit ist die praktische Übung abgeschlossen.

► Lösung

Veröffentlichen von Enterprise Container-Images

In dieser praktischen Übung veröffentlichen Sie ein in OCI formatiertes Container-Image in einer externen Registry und stellen eine Anwendung aus diesem Image mit einem Image-Stream bereit.



Anmerkung

Für den am Ende jeder praktischen Übung verwendeten Befehl `grade` ist es erforderlich, dass Sie die exakten Projektnamen und anderen Identifikatoren, wie in der Spezifikation der praktischen Übung angegeben, verwenden.

Ergebnisse

Sie sollten in der Lage sein, eine Anwendung aus einem in einer externen Registry gespeicherten Image zu erstellen.

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen aktiven OpenShift-Cluster
- Auf OCI-konforme Dateien für das Beispiel-Container-Image (`php-info`)

Führen Sie den folgenden Befehl auf der `workstation`-VM aus, um die Voraussetzungen zu überprüfen und die Lösungsdateien herunterzuladen:

```
[student@workstation ~]$ lab expose-image start
```

Anforderungen

Das Anwendungs-Image umfasst eine PHP-Anwendung, die die Webserverumgebung und die PHP-Interpreter-Konfiguration anzeigt. Stellen Sie die Anwendung wie folgt bereit:

- Hosten Sie den Image-Stream für das außerhalb von OpenShift erstellt Image in einem Projekt mit dem Namen `youruser-common`.

Der Name des Container-Images lautet `php-info`. Die OCI-formatierten Image-Ebenen und das Manifest befinden sich im Ordner `/home/student/D0288/labs/expose-image/php-info`. Übertragen Sie dieses Image in ein privates Image-Repository in Ihrem Quay.io-Benutzerkonto.

- Stellen Sie die Anwendung in einem Projekt namens `youruser-expose-image` bereit.

Die Ressourcen der Anwendung werden als `info` bezeichnet. Greifen Sie auf die Anwendung mit dem standardmäßigen Hostnamen zu, der von OpenShift zugewiesen wurde.

Kapitel 3 | Veröffentlichen von Enterprise Container-Images

- Verwenden Sie die Konfigurationsdatei `/usr/local/etc/ocp4.config`, um Kursraumkonfigurationsdaten wie die Master-API-URL des OpenShift-Clusters abzurufen.

Anweisungen

1. Übertragen Sie das in OCI formatierte `php-info`-Container-Image an Quay.io.
 - 1.1. Verifizieren Sie, ob der Ordner `php-info` ein in OCI formatiertes Container-Image enthält:

```
[student@workstation ~]$ ls ~/D0288/labs/expose-image/php-info  
blobs index.json oci-layout
```

- 1.2. Laden Sie die Konfiguration Ihrer Kursumgebung.

Führen Sie den folgenden Befehl aus, um die in der ersten angeleiteten Übung erstellten Konfigurationsvariablen zu laden:

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

- 1.3. Melden Sie sich mit dem Befehl `podman` bei Quay.io an.

```
[student@workstation ~]$ podman login -u ${RHT_OCP4_QUAY_USER} quay.io  
Password:  
Login Succeeded!
```

- 1.4. Übertragen Sie das in OCI formatierte Container-Image mit dem Befehl `skopeo` an Quay.io. Sie können den Befehl `skopeo` aus dem Skript `push-image.sh` im Ordner `/home/student/D0288/labs/expose-image` kopieren oder ausführen.

```
[student@workstation ~]$ skopeo copy \  
oci:/home/student/D0288/labs/expose-image/php-info \  
docker://quay.io/${RHT_OCP4_QUAY_USER}/php-info  
...output omitted...  
Writing manifest to image destination  
Storing signatures
```

- 1.5. Untersuchen Sie mit `Skopeo` das Image in der externen Registry, um zu überprüfen, ob es mit `latest` gekennzeichnet ist.

```
[student@workstation ~]$ skopeo inspect \  
docker://quay.io/${RHT_OCP4_QUAY_USER}/php-info  
{  
  "Name": "quay.io/yourquayuser/php-info",  
  "Tag": "latest",  
  ...output omitted...
```

2. Erstellen Sie als Entwicklerbenutzer das Projekt `youruser-common`, um den Image-Stream zu hosten, der auf das Image in der externen Registry zeigt. Erstellen Sie ein Secret mit Anmelddaten für Quay.io und erstellen Sie den `php-info`-Image-Stream.

Kapitel 3 | Veröffentlichen von Enterprise Container-Images

Erstellen Sie den Image-Stream mit der Option `--reference-policy local`, damit auch andere Projekte, die diesen Image-Stream verwenden, das im Projekt `youruser-common` gespeicherte Secret verwenden können.

- 2.1. Melden Sie sich bei OpenShift mit Ihrem Entwicklerbenutzerkonto an:

```
[student@workstation D0288-apps]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful
...output omitted...
```

- 2.2. Erstellen Sie ein Projekt, um den Image-Stream zu hosten.

```
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-common
Now using project "youruser-common" on server
"https://api.cluster.domain.example.com:6443".
```

- 2.3. Erstellen Sie ein Secret aus dem Zugriffstoken der Container-Registry-API, das von Podman gespeichert wurde.

Sie können den Befehl `oc create secret` aus dem Skript `create-secret.sh` im Ordner `/home/student/D0288/labs/expose-image` kopieren oder ausführen.

```
[student@workstation ~]$ oc create secret generic quayio \
--from-file .dockerconfigjson=${XDG_RUNTIME_DIR}/containers/auth.json \
--type kubernetes.io/dockerconfigjson
secret/quayio created
```

- 2.4. Importieren Sie mit dem Befehl `oc import-image` das Container-Image:

```
[student@workstation ~]$ oc import-image php-info --confirm \
--reference-policy local \
--from quay.io/${RHT_OCP4_QUAY_USER}/php-info
imagestream.image.openshift.io/php-info imported
...output omitted...
latest
tagged from quay.io/youruser/php-info
will use insecure HTTPS or HTTP connections

* quay.io/youruser/php-info@sha256:4366...f937
  Less than a second ago
...output omitted...
```

- 2.5. Überprüfen Sie, ob ein Image-Stream-Tag erstellt wurde und ob es Metadaten über das Container-Image `php-info` enthält:

```
[student@workstation ~]$ oc get istag
NAME          IMAGE REF   ...
php-info:latest  image-registry.openshift-image-registry.svc:5000/youruser-
common/php-info@sha256:4366...f937  ...
```

Kapitel 3 | Veröffentlichen von Enterprise Container-Images

3. Erstellen Sie ein neues Projekt mit dem Namen `youruser-expose-image`. Erstellen Sie anschließend eine neue Anwendung, indem Sie das Container-Image aus dem Image-Stream bereitstellen.

Gewähren Sie die Rolle `system:image-puller` im Projekt `youruser-common` allen Servicekonten im Projekt `youruser-expose-image`. Mit dieser Rolle können Pods aus einem Projekt Image-Streams aus einem anderen Projekt verwenden. Das Skript `grant-puller-role.sh` im Ordner `/home/student/D0288/labs/expose-image` enthält den Befehl `oc policy`, der diesen Vorgang ausführt.

- 3.1. Erstellen Sie ein Projekt zum Hosten der Anwendung:

```
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-expose-image
```

- 3.2. Gewähren Sie Servicekonten aus dem neuen Projekt `youruser-expose-image` Zugriff auf die Image-Streams aus dem Projekt `youruser-common`. Sie können den folgenden `oc policy`-Befehl aus dem Skript `grant-puller-role.sh` im Ordner `/home/student/D0288/labs/expose-image` kopieren oder ausführen.

```
[student@workstation ~]$ oc policy add-role-to-group \
-n ${RHT_OCP4_DEV_USER}-common system:image-puller \
system:serviceaccounts:${RHT_OCP4_DEV_USER}-expose-image
clusterrole.rbac.authorization.k8s.io/system:image-puller added:
"system:serviceaccounts:youruser-expose-image"
```

- 3.3. Stellen Sie die Testanwendung über den Image-Stream im Projekt `common` bereit:

```
[student@workstation ~]$ oc new-app --name info \
-i ${RHT_OCP4_DEV_USER}-common/php-info
...output omitted...
--> Creating resources ...
imagestreamtag.image.openshift.io "info:latest" created
deployment.apps "info" created
service "info" created
--> Success
...output omitted...
```

- 3.4. Warten Sie, bis der Anwendungs-Pod bereit ist und ausgeführt wird:

```
[student@workstation ~]$ oc get pod
NAME           READY   STATUS    RESTARTS   AGE
info-5c687bc4bc-j4sdz  1/1     Running   0          26s
```

4. Stellen Sie die Anwendung bereit und testen Sie sie. Überprüfen Sie, ob die Anwendung die PHP-Interpreter-Konfigurationsseite zurückgibt.

- 4.1. Stellen Sie die Anwendung `info` zur Verfügung:

```
[student@workstation ~]$ oc expose svc info
route.route.openshift.io/info exposed
```

- 4.2. Rufen Sie den Hostnamen ab, den OpenShift der Route zuweist:

```
[student@workstation ~]$ oc get route info
NAME      HOST/PORT
info      info-youruser-expose-image.apps.cluster.domain.example.com
```

- 4.3. Testen Sie die Anwendung mit dem Befehl `curl` und der Route aus dem vorherigen Schritt. Alternativ können Sie einen Webbrower verwenden.

```
[student@workstation ~]$ curl \
http://info-${RHT_OCP4_DEV_USER}-expose-image.${RHT_OCP4_WILDCARD_DOMAIN}
...output omitted...
<title>phpinfo()</title><meta name="ROBOTS" content="NOINDEX, NOFOLLOW, NOARCHIVE" />
</head>
...output omitted...
<tr><td class="e">Server API </td><td class="v">FPM/FastCGI </td></tr>
...output omitted...
<tr><td class="e">PHP API </td><td class="v">20170718 </td></tr>
...output omitted...
```

5. Bewerten Sie Ihre Arbeit.

Führen Sie den folgenden Befehl auf der `workstation`-VM aus, um zu überprüfen, ob alle Aufgaben abgeschlossen wurden:

```
[student@workstation ~]$ lab expose-image grade
```

6. Löschen Sie die OpenShift-Projekte und entfernen Sie das Image-Repository aus Quay.io.

- 6.1. Löschen Sie das Projekt `youruser-expose-image`:

```
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-expose-image
project.project.openshift.io "youruser-expose-image" deleted
```

- 6.2. Löschen Sie das Projekt `youruser-common`:

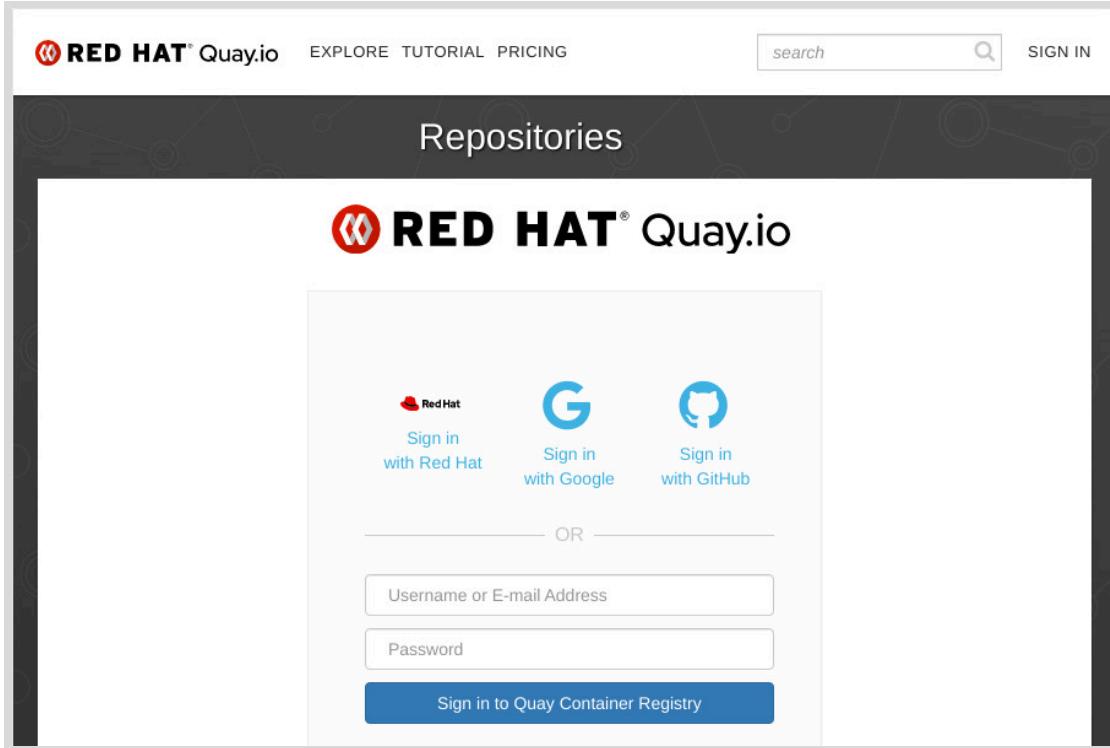
```
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-common
project.project.openshift.io "youruser-common" deleted
```

- 6.3. Löschen Sie das Container-Image aus der externen Registry:

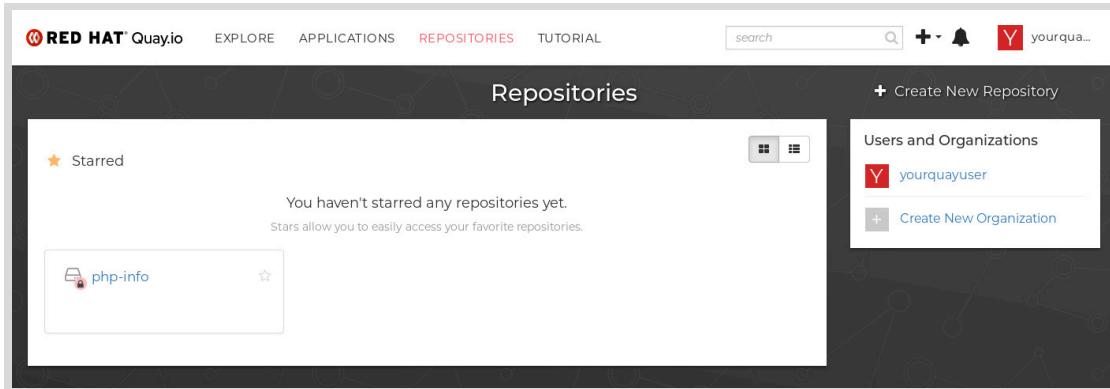
```
[student@workstation ~]$ skopeo delete \
docker://quay.io/${RHT_OCP4_QUAY_USER}/php-info:latest
```

- 6.4. Melden Sie sich bei Quay.io mit Ihrem persönlichen kostenlosen Benutzerkonto an.

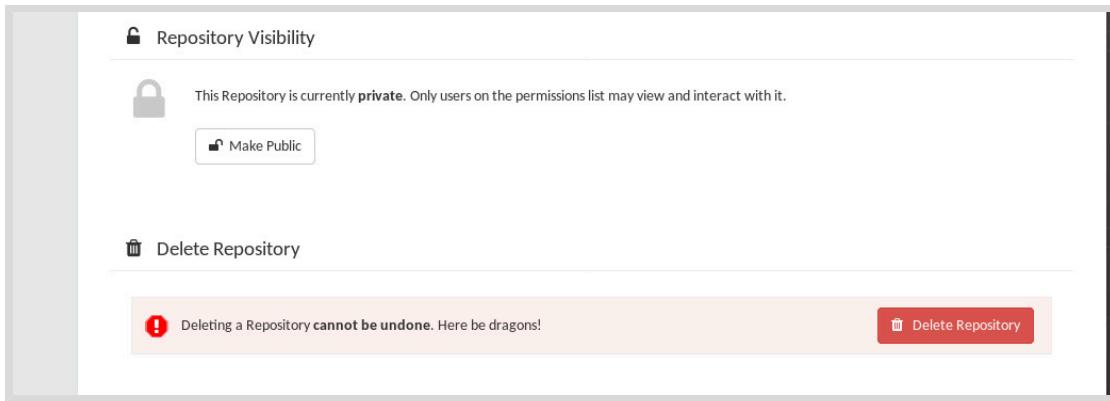
Navigieren Sie zu <http://quay.io> und klicken Sie auf **Sign In**, um Ihre Benutzeranmeldedaten einzugeben. Klicken Sie auf **Sign in to Quay Container Registry**, um sich bei Quay.io anzumelden.



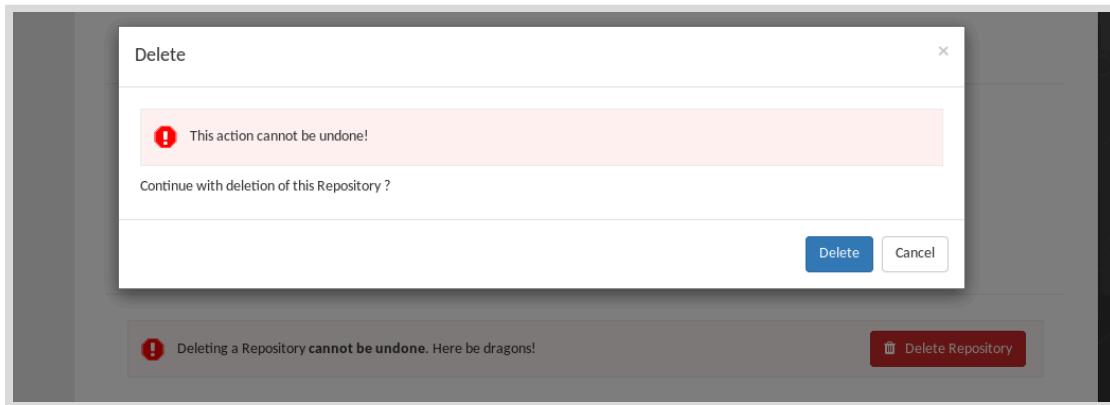
- 6.5. Klicken Sie im Hauptmenü von Quay.io auf **Repositories** und suchen Sie nach **php-info**. Klicken Sie auf **php-info**, um die Seite **Repository Activity** zu öffnen.



- 6.6. Scrollen Sie auf der Seite **Repository Activity** für das **php-info**-Repository nach unten und klicken Sie auf das Zahnradsymbol, um den den **Settings** -Tab anzulegen. Scrollen Sie nach unten und klicken Sie auf **Delete Repository**.



- 6.7. Klicken Sie im Dialogfeld **Delete** auf **Delete**, um das Löschen des `php-info`-Repository zu bestätigen. Nach einigen Augenblicken werden Sie zur Seite **Repositories** zurückgeleitet. Sie können sich jetzt bei Quay.io abmelden.



Beenden

Führen Sie auf der `workstation`-VM den Befehl `lab expose-image finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken. Die Aktion „Beenden“ veröffentlicht dieses Projekt und seine Ressourcen.

```
[student@workstation ~]$ lab expose-image finish
```

Hiermit ist die praktische Übung abgeschlossen.

Zusammenfassung

In diesem Kapitel wurden die folgenden Themen behandelt:

- OpenShift-Entwickler interagieren mit vielen Arten von Registry-Servern, für die möglicherweise eine Authentifizierung erforderlich ist, einschließlich der internen Registry von OpenShift.
- OpenShift erfordert, dass Entwickler Secrets erstellen, um Zugriffstoken in privaten, externen Registrys zu speichern.
- Die Linux-Container-Tools (Skopeo, Podman und Buildah) können wie jeder andere Registry-Server auf die interne OpenShift-Registry zugreifen, indem sie das OpenShift-Benutzerzugriffstoken als sichere oder unsichere Registry verwenden.
- Image-Streams und Image-Stream-Tag-Ressourcen von OpenShift bieten stabile Verweise auf Container-Images und trennen Entwickler zudem von Registry-Serveradressen.

Kapitel 4

Verwalten von Builds auf OpenShift

Ziel

Beschreiben des OpenShift-Build-Prozesses und der Trigger sowie Verwalten von Builds

Ziele

- Beschreiben des OpenShift-Build-Prozesses
- Verwalten von Anwendungs-Builds mithilfe der BuildConfig-Ressource und CLI-Befehle
- Auslösen des Build-Prozesses anhand unterstützter Methoden
- Bearbeiten der Logik im Anschluss an die Erstellung mithilfe des Build-Hook „post-commit“

Abschnitte

- Beschreiben des OpenShift-Build-Prozesses (und Test)
- Verwalten von Anwendungs-Builds (und angeleitete Übung)
- Auslösen von Builds (und angeleitete Übung)
- Implementieren des Build-Hook „post-commit“ (und angeleitete Übung)

Praktische Übung

Verwalten von Builds auf OpenShift

Beschreiben des Red Hat OpenShift-Build-Prozesses

Ziele

Nach Abschluss dieses Abschnitts sollten Sie in der Lage sein, den Build-Prozess von Red Hat OpenShift zu beschreiben.

Der Build-Prozess

Der Build-Prozess von Red Hat OpenShift Container Platform wandelt entweder Quellcode oder Binärdateien und andere Eingabeparameter in Container-Images um, die für die Bereitstellung auf der Plattform verfügbar werden. Red Hat OpenShift benötigt zur Erstellung eines Container-Images eine `BuildConfig`-Ressource. Die Konfiguration dieser Ressource umfasst eine Build-Strategie und eine oder mehrere Eingabequellen, wie Git, Binärdateien oder Inline-Definitionen.

Build-Strategien

Nachfolgend werden die in Red Hat OpenShift verfügbaren Build-Strategien aufgeführt:

- Source-to-Image-Build (S2I)
- Docker-Build
- Benutzerdefinierter Build

Alle Strategien erfordern ein Container-Image.

Source-to-Image-Build (S2I)

Bei der Source-to-Image-Strategie wird, basierend auf einem Anwendungsquellcode oder Anwendungsbinärdateien, ein neues Container-Image erstellt. Red Hat OpenShift klonst den Anwendungsquellcode oder kopiert die Anwendungsbinärdateien in ein kompatibles Builder-Image und stellt ein neues Container-Image zusammen, das auf der Plattform bereitgestellt werden kann.

Diese Strategie erleichtert Entwicklern die Erstellung von Container-Images, da sie mit den ihnen vertrauten Tools umgesetzt werden kann, anstatt Betriebssystembefehle auf niedriger Ebene wie `yum` in `Containerfiles` ausführen zu müssen.

Die Source-to-Image-Strategie von Red Hat OpenShift basiert auf dem S2I-Prozess (Source-to-Image).

Docker-Build

Die `docker build`-Strategie verwendet den Befehl `buildah`, um ein neues Container-Image zu erstellen, wenn eine `Containerfile`-Datei vorhanden ist. Die `docker`-Strategie kann die `Containerfile`-Datei und die Artefakte abrufen, um das Container-Image aus einem Git-Repository zu erstellen, oder kann eine inline in der Build-Konfiguration bereitgestellte `Containerfile`-Datei als Build-Quelle verwenden.

Der Docker-Build wird als Pod im Red Hat OpenShift-Cluster ausgeführt. Entwickler benötigen keine Docker-Tools auf ihrer Workstation.



Anmerkung

Docker-Builds setzen erweiterte Berechtigungen voraus. Der Red Hat OpenShift-Cluster-Administrator kann einigen oder auch allen Benutzern die Berechtigung zum Starten von Docker-Builds verweigern.

Benutzerdefinierter Build

Bei der `custom build`-Strategie wird ein Builder-Image für den Build-Prozess festgelegt. Diese Strategie ermöglicht Entwicklern die Anpassung des Build-Prozesses. Weitere Informationen zur Erstellung eines benutzerdefinierten Builder-Images finden Sie im Abschnitt zu den Referenzen für diese Einheit.

Build-Eingabequellen

Eine Build-Eingabequelle stellt Quellinhalte für Builds bereit. Red Hat OpenShift unterstützt die folgenden sechs Arten von Eingabequellen, die nach Rangfolge aufgeführt sind:

- `Containerfile`: Gibt das Containerfile zur Erstellung eines Images in einer Zeile an.
- `Image`: Sie können für den Build-Prozess weitere Dateien bereitstellen, wenn Sie den Build aus Images erstellen.
- `Git`: Red Hat OpenShift klonnt den Quellcode der Eingabeanwendung aus einem Git-Repository. Sie haben die Möglichkeit, den Standardspeicherort im Repository zu konfigurieren, in dem der Build nach dem Anwendungsquellcode sucht.
- `Binary`: Ermöglicht das Streamen von binären Inhalten aus einem lokalen Dateisystem in den Builder.
- `Input secrets`: Sie können EingabeSecrets verwenden, um das Erstellen von Anmeldedaten für den Build zuzulassen, die im endgültigen Anwendungs-Image nicht verfügbar sind.
- `External artifacts`: Ermöglicht das Kopieren von binären Dateien in den Build-Prozess.

Sie können mehrere Eingaben in einem einzigen Build kombinieren. Da jedoch das Inline-Containerfile Vorrang hat, überschreibt es alle anderen Containerfiles, die von einer anderen Eingabe bereitgestellt werden. Darüber hinaus schließen sich binäre Eingaben und Git-Repositories gegenseitig aus.



Anmerkung

Obwohl Red Hat OpenShift zahlreiche Strategien und Eingabequellen bietet, werden in den meisten Szenarien die Strategien `Source` oder `Docker` mit einem Git-Repository als Eingabequelle eingesetzt.

BuildConfig-Ressource

Die Build-Konfiguration legt den Ablauf des Build-Prozesses fest. Die `BuildConfig`-Ressource definiert eine einzelne Build-Konfiguration und eine Reihe von Triggern zur Erstellung eines neuen Builds durch Red Hat OpenShift.

Red Hat OpenShift generiert die `BuildConfig`-Ressource mit dem Befehl `oc new-app`. Die Ressource kann auch mithilfe der Schaltfläche **Add to Project** über die Web Console oder durch Erstellen einer Anwendung aus einer Vorlage generiert werden.

Kapitel 4 | Verwalten von Builds auf OpenShift

Im folgenden Beispiel wird eine PHP-Anwendung unter Verwendung der Source-Strategie und der Eingabequelle Git erstellt:

```
{  
    "kind": "BuildConfig",  
    "apiVersion": "v1",  
    "metadata": {  
        "name": "php-example", ①  
        ...output omitted...  
    },  
    "spec": {  
        "triggers": [ ②  
            {  
                "type": "GitHub",  
                "github": {  
                    "secret": "gukAWHzq1On4AJlMjvjS" ③  
                }  
            },  
            ...output omitted...  
        ],  
        "runPolicy": "Serial", ④  
        "source": { ⑤  
            "type": "Git",  
            "git": {  
                "uri": "http://services.lab.example.com/php-helloworld"  
            }  
        },  
        "strategy": { ⑥  
            "type": "Source",  
            "sourceStrategy": {  
                "from": {  
                    "kind": "ImageStreamTag",  
                    "namespace": "openshift",  
                    "name": "php:7.0"  
                }  
            }  
        },  
        "output": { ⑦  
            "to": {  
                "kind": "ImageStreamTag",  
                "name": "php-example:latest"  
            }  
        },  
        ...output omitted...  
    },  
    ...output omitted...  
}
```

- ① Definiert eine neue BuildConfig-Ressource mit dem Namen php-example.
- ② Definiert die Trigger zum Starten neuer Builds.

- ③ Autorisierungs-String für den Webhook; wird in Red Hat OpenShift nach dem Zufallsprinzip generiert. Externe Anwendungen senden diese Zeichenfolge als Teil der Webhook-URL, um neue Builds auszulösen.
- ④ Das Attribut `runPolicy` definiert, ob Builds gleichzeitig gestartet werden können. Der Wert `Serial` gibt an, dass eine gleichzeitige Erstellung nicht möglich ist.
- ⑤ Das Attribut `source` definiert die Eingabequelle des Builds. In diesem Beispiel wird ein Git-Repository verwendet.
- ⑥ Definiert die Build-Strategie zur Erstellung des endgültigen Container-Images. In diesem Beispiel wird die `Source`-Strategie verwendet.
- ⑦ Das Attribut `output` legt fest, wohin das neue Container-Image nach der Erstellung übertragen werden soll.



Literaturhinweise

Weitere Informationen zu benutzerdefinierten Build-Images und Build-Eingabequellen finden Sie im Kapitel *Creating Build Inputs* des Handbuchs *Builds* für Red Hat OpenShift Container Platform 4.6 unter
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/builds/creating-build-inputs

Source- im Vergleich zu binären S2I-Builds werden näher erläutert unter
<https://developers.redhat.com/blog/2018/09/26/source-versus-binary-s2i-workflows-with-red-hat-openshift-application-runtimes/>

► Quiz

Der OpenShift-Build-Prozess

Wählen Sie die richtige(n) Antwort(en) auf die folgenden Fragen aus:

► 1. Welche drei der folgenden Optionen sind zulässige Strategien zur Erstellung eines Container-Images? (Wählen Sie drei Antworten aus.)

- a. Docker
- b. Git
- c. Source-to-Image
- d. Custom

► 2. Welche zwei der folgenden Optionen sind zulässige Eingabequellentypen zur Erstellung eines Container-Images? (Wählen Sie zwei Antworten aus.)

- a. Git
- b. SVN
- c. Filesystem
- d. Containerfile

► 3. Welche drei der folgenden Aussagen treffen im Hinblick auf die BuildConfig-Ressource in der nachfolgenden Darstellung zu? (Wählen Sie drei Antworten aus.)

```
{  
    "kind": "BuildConfig",  
    "apiVersion": "v1",  
    "metadata": {  
        "name": "php-example",  
    },  
    "spec": {  
        ...  
        "runPolicy": "Serial",  
        "source": {  
            "type": "Git",  
            "git": {  
                "uri": "http://services.lab.example.com/php-helloworld"  
            }  
        },  
        "strategy": {  
            "type": "Source",  
            "sourceStrategy": {  
                "from": {  
                    "kind": "ImageStreamTag",  
                    "namespace": "openshift",  
                    "name": "php:7.0"  
                }  
            }  
        },  
        "output": {  
            "to": {  
                "kind": "ImageStreamTag",  
                "name": "php-example:latest"  
            }  
        },  
        ...  
    },  
}
```

- 4. Welche der folgenden Eingabequellen wird bei Einsatz der Docker-Strategie in einer Build-Konfiguration verwendet, die mit dem Befehl `oc new-app` erzeugt wurde?
- a. Containerfile
 - b. Binary
 - c. Git
 - d. Nichts davon
- 5. Ein Entwickler möchte unter Einsatz der Source-Strategie eine PHP-Anwendung erstellen. Welche der folgenden Optionen ermöglicht die Konfiguration der Build-Konfiguration zur Verwendung des S2I-Builder-Images für eine PHP-Anwendung?
- a. Verwendung des Attributs `from` für die Source-Strategie
 - b. Verwendung des Attributs `from` aus der Image-Eingabequelle
 - c. Verwendung des Attributs `from` aus der Containerfile-Eingabequelle
 - d. Die Konfiguration des S2I-Builder-Images ist nicht erforderlich. Während der Erstellung werden im Rahmen des S2I-Prozesses die Quellsprache erkannt und automatisch ein PHP-Builder-Image ausgewählt.
 - e. Nichts davon.

► Lösung

Der OpenShift-Build-Prozess

Wählen Sie die richtige(n) Antwort(en) auf die folgenden Fragen aus:

- ▶ 1. Welche drei der folgenden Optionen sind zulässige Strategien zur Erstellung eines Container-Images? (Wählen Sie drei Antworten aus.)
 - a. Docker
 - b. Git
 - c. Source-to-Image
 - d. Custom

- ▶ 2. Welche zwei der folgenden Optionen sind zulässige Eingabequellentypen zur Erstellung eines Container-Images? (Wählen Sie zwei Antworten aus.)
 - a. Git
 - b. SVN
 - c. Filesystem
 - d. Containerfile

► 3. Welche drei der folgenden Aussagen treffen im Hinblick auf die BuildConfig-Ressource in der nachfolgenden Darstellung zu? (Wählen Sie drei Antworten aus.)

```
{  
    "kind": "BuildConfig",  
    "apiVersion": "v1",  
    "metadata": {  
        "name": "php-example",  
    },  
    "spec": {  
        ...  
        "runPolicy": "Serial",  
        "source": {  
            "type": "Git",  
            "git": {  
                "uri": "http://services.lab.example.com/php-helloworld"  
            }  
        },  
        "strategy": {  
            "type": "Source",  
            "sourceStrategy": {  
                "from": {  
                    "kind": "ImageStreamTag",  
                    "namespace": "openshift",  
                    "name": "php:7.0"  
                }  
            }  
        },  
        "output": {  
            "to": {  
                "kind": "ImageStreamTag",  
                "name": "php-example:latest"  
            }  
        },  
        ...  
    },  
}
```

► 4. Welche der folgenden Eingabequellen wird bei Einsatz der Docker-Strategie in einer Build-Konfiguration verwendet, die mit dem Befehl `oc new-app` erzeugt wurde?

- a. Containerfile
- b. Binary
- c. Git
- d. Nichts davon

► 5. Ein Entwickler möchte unter Einsatz der Source-Strategie eine PHP-Anwendung erstellen. Welche der folgenden Optionen ermöglicht die Konfiguration der Build-Konfiguration zur Verwendung des S2I-Builder-Images für eine PHP-Anwendung?

- a. Verwendung des Attributs `from` für die Source-Strategie
- b. Verwendung des Attributs `from` aus der Image-Eingabequelle
- c. Verwendung des Attributs `from` aus der Containerfile-Eingabequelle
- d. Die Konfiguration des S2I-Builder-Images ist nicht erforderlich. Während der Erstellung werden im Rahmen des S2I-Prozesses die Quellsprache erkannt und automatisch ein PHP-Builder-Image ausgewählt.
- e. Nichts davon.

Verwalten von Anwendungs-Builds

Ziele

Nach Abschluss dieses Abschnitts sollten Sie in der Lage sein, Anwendungs-Builds unter Verwendung der Build-Konfigurationsressource und der CLI-Befehle zu verwalten.

Erstellen einer Build-Konfiguration

In Red Hat OpenShift Container Platform werden Builds über die Build-Konfigurationsressource verwaltet. Eine Build-Konfiguration kann auf zwei Arten erstellt werden:

Verwenden des Befehls `oc new-app`

Mit diesem Befehl werden Build-Konfigurationen den festgelegten Argumenten entsprechend erstellt. Beispiel: Bei der Definition eines Git-Repository wird im Anschluss eine Build-Konfiguration unter Einsatz der Source-Strategie erstellt. Wenn das Argument eine Vorlage ist und für die Vorlage eine Build-Konfiguration definiert wurde, wird eine Build-Konfiguration anhand der Vorlagenparameter erstellt.

Verwenden einer benutzerdefinierten Build-Konfiguration

Erstellen Sie eine benutzerdefinierte Build-Konfiguration unter Verwendung der YAML- oder JSON-Syntax, und importieren Sie diese anschließend mit dem Befehl `oc create -f` in Red Hat OpenShift.

Verwalten von Anwendungs-Builds mit der CLI

Red Hat OpenShift stellt mehrere Optionen bereit, die Entwicklern die Verwaltung von Anwendungs-Builds und Build-Konfigurationen über die CLI ermöglichen. Diese Befehle werden verwendet, um den Lifecycle einer Anwendung zu verwalten, insbesondere während der Entwicklung. Beachten Sie, dass die Build-Konfiguration im Vergleich zur Bereitstellung eine separate Phase ist. Ein erfolgreicher Build in Red Hat OpenShift führt zu einem neuen Container-Image, das eine neue Bereitstellung auslöst.

`oc start-build`

Startet einen neuen Build manuell. Der Name der Build-Konfigurationsressource ist das einzige erforderliche Argument zum Starten eines neuen Builds.

```
[user@host ~]$ oc start-build name
```

Ein erfolgreicher Build erstellt ein neues Container-Image in der Ausgabe ImageStreamTag. Wenn eine Deploymentkonfiguration einen Trigger auf diesem ImageStreamTag definiert, wird der Deploymentprozess gestartet.

`oc cancel-build`

Bricht den Build ab. Wenn ein Build beispielsweise mit einer falschen Anwendungsversion gestartet wurde und die Anwendung nicht bereitgestellt werden kann, können Sie den Build abbrechen, bevor er fehlschlägt.

Es können nur Builds abgebrochen werden, die sich im Status „Running“ oder „Pending“ befinden. Beim Abbrechen eines Builds wird der Build-Pod beendet. In diesem Fall ist kein

Kapitel 4 | Verwalten von Builds auf OpenShift

neues Container-Image für die Weiterleitung verfügbar. Eine Bereitstellung wird daher nicht ausgelöst.

Geben Sie den Namen der Build-Konfigurationsressource an, um den Build abzubrechen:

```
[user@host ~]$ oc cancel-build name
```

oc delete

Löscht eine Build-Konfiguration. Eine Build-Konfiguration muss üblicherweise gelöscht werden, wenn Sie eine neue Build-Konfiguration aus einer Datei importieren müssen. Denken Sie daran, dass bc eine Kurznotation für die Build-Konfiguration ist.

```
[user@host ~]$ oc delete bc/name
```

Mit dem folgenden Befehl wird ein Build gelöscht (keine Build-Konfiguration), um den vom Build belegten Speicherplatz wieder freizugeben. Eine Build-Konfiguration kann über mehrere Builds verfügen. Geben Sie den Build-Namen an, um den Build zu löschen:

```
[user@host ~]$ oc delete build/name-1
```

oc describe

Gibt Details zu Build-Konfigurationsressourcen und den zugeordneten Builds an; beispielsweise Informationen zu Bezeichnungen, der Strategie, Webhooks usw.

```
[user@host ~]$ oc describe bc name
```

Beschreiben Sie ein Build durch Angabe des Build-Namens:

```
[user@host ~]$ oc describe build name-1
```

oc logs

Zeigt die Build-Logs an. Sie können überprüfen, ob Ihre Anwendung ordnungsgemäß erstellt wird. Bei Ausführung dieses Befehls werden auch Logs aus dem fertiggestellten Build angezeigt. Logs aus gelöschten oder bereinigten Builds können nicht überprüft werden. Build-Logs können auf zwei Arten angezeigt werden:

- Anzeigen der Build-Logs für das zuletzt erstellte Build.

```
[user@host ~]$ oc logs -f bc/name
```

Die Option -f folgt dem Log, bis der Befehl mit Strg+C beendet wird.

- Zeigt die Build-Logs für einen spezifischen Build an:

```
[user@host ~]$ oc logs build/name-1
```

Bereinigen von Builds

Standardmäßig werden die fünf aktuellen abgeschlossenen Builds beibehalten. Sie können die Anzahl der vorherigen Builds, die beibehalten werden sollen, mit dem Attribut `successfulBuildsHistoryLimit` und dem Attribut `failedBuildsHistoryLimit` begrenzen, wie im folgenden YAML-Codeausschnitt einer Build-Konfiguration gezeigt:

```
apiVersion: "v1"
kind: "BuildConfig"
metadata:
  name: "sample-build"
spec:
  successfulBuildsHistoryLimit: 2
  failedBuildsHistoryLimit: 2
  ...contents omitted...
```

Zu den fehlgeschlagenen Builds gehören Builds mit dem Status `failed`, `canceled` oder `error`. Builds werden nach ihrer Erstellungszeit sortiert, wobei die ältesten Builds zuerst bereinigt werden.



Anmerkung

Red Hat OpenShift-Administratoren können Builds manuell mit dem Befehl `oc adm` zum Bereinigen von Objekten bereinigen.

Logausführlichkeit

Red Hat OpenShift bietet zwei verschiedene Mechanismen zum Konfigurieren der Logausführlichkeit von Builds. Die erste Konfiguration ist global und ein Administrator kann die Logausführlichkeit von Builds für den gesamten Cluster definieren. Die zweite betrifft nur die Ausführlichkeit der Build-Logs der Builds aus einer bestimmten Build-Konfiguration. Dies bedeutet, dass Entwickler weiterhin die Möglichkeit haben, die globale Konfiguration mit einer spezifischeren Build-Konfiguration zu überschreiben, wenn sie eine andere Ebene der Logausführlichkeit benötigen.

Um die Standardeinstellung des Administrators zu überschreiben, kann ein Entwickler die Build-Konfigurationsressource bearbeiten und die Umgebungsvariable `BUILD_LOGLEVEL` als Bestandteil der Source-Strategie bzw. Docker-Strategie hinzufügen, um einen spezifischen Log-Level zu konfigurieren:

```
[user@host ~]$ oc set env bc/name BUILD_LOGLEVEL="4"
```

Der Wert muss eine Zahl zwischen null und fünf sein. Null ist der Standardwert; bei Auswahl dieses Werts werden weniger als fünf Logs angezeigt. Wenn Sie die Anzahl erhöhen, ist die Logierung von Meldungen ausführlicher und die Logs enthalten mehr Details.

Bereinigen von Builds

Bei der „Bereinigung“ eines Builds wird ein abgeschlossener oder fehlgeschlagener Build gelöscht. Red Hat OpenShift kann diesen Vorgang der Konfiguration entsprechend automatisch durchführen. Alternativ können Sie einen Build mit dem Befehl `oc delete` bzw. `oc adm prune` manuell bereinigen.

Administratoren können Builds bereinigen und Festplattenspeicher freigeben, um eine Anhäufung von Build-Versionen im etcd-Datenspeicher zu vermeiden. Es gibt verschiedene Konfigurationsoptionen zur Bereinigung von Builds. So können beispielsweise verwaiste Objekte bereinigt werden, Bereinigungen auf Grundlage der Anzahl erfolgreich ausgeführter Builds bzw. auf Grundlage der Anzahl fehlgeschlagener Builds und vieles mehr durchgeführt werden.

In manchen Fällen müssen Sie einen Administrator auffordern, die Bereinigungskonfiguration zu ändern, damit Build-Logs zur Behebung von Build-Problemen länger aufbewahrt werden, als üblicherweise vorgesehen. Diese Themen sind für den aktuellen Kurs nicht vorgesehen.



Literaturhinweise

Weitere Informationen über die Verwaltung von Anwendungs-Builds finden Sie im Kapitel *Performing Basic Builds* des Handbuchs *Builds* für Red Hat OpenShift Container Platform 4.6 unter

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/builds/basic-build-operations

► Angeleitete Übung

Verwalten von Anwendungs-Builds

In dieser Übung verwalten Sie Anwendungs-Builds mit OpenShift unter Verwendung der Source-Strategie und einer Git-Eingabequelle.

Ergebnisse

Sie sollten in der Lage sein, die CLI zum Verwalten von Builds in OpenShift zu verwenden.

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen aktiven OpenShift-Cluster
- Auf das OpenJDK 1.8-S2I-Builder-Image und den Image-Stream (`redhat-openjdk18-openshift:1.8`)
- Auf die Beispielanwendung im Git-Repository, im Verzeichnis `java-serverhost`

Führen Sie den folgenden Befehl auf der `workstation`-VM aus, um die Voraussetzungen zu überprüfen und die Dateien herunterzuladen, die zum Durchführen dieser Übung erforderlich sind:

```
[student@workstation ~]$ lab manage-builds start
```

Anweisungen

► 1. Untersuchen Sie den Java-Quellcode für die Beispielanwendung.

- 1.1. Wechseln Sie zu Ihrem lokalen Klon des D0288-apps-Git-Repository und checken Sie den `main`-Branch des Kurs-Repository aus, um sicherzustellen, dass Sie diese Übung von einem als fehlerfrei bekannten Zustand aus starten:

```
[student@workstation ~]$ cd D0288-apps
[student@workstation D0288-apps]$ git checkout main
...output omitted...
```

- 1.2. Erstellen Sie einen neuen Branch, um alle Änderungen zu speichern, die Sie während dieser Übung vornehmen:

```
[student@workstation D0288-apps]$ git checkout -b manage-builds
Switched to a new branch 'manage-builds'
[student@workstation D0288-apps]$ git push -u origin manage-builds
...output omitted...
* [new branch]      manage-builds -> manage-builds
Branch manage-builds set up to track remote branch manage-builds from origin.
```

- 1.3. Überprüfen Sie den Java-Quellcode für die Anwendung im Ordner `java-serverhost`.

Untersuchen Sie die Datei `/home/student/D0288-apps/java-serverhost/src/main/java/com/redhat/training/example/javaserverhost/rest/ServerHostEndPoint.java`:

```
package com.redhat.training.example.javaserverhost.rest;

import javax.ws.rs.Path;
import javax.ws.rs.core.Response;
import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import java.net.InetAddress;

@Path("/")
public class ServerHostEndPoint {

    @GET
    @Produces("text/plain")
    public Response doGet() {
        String host = "";
        try {
            host = InetAddress.getLocalHost().getHostName();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        String msg = "I am running on server " + host + " Version 1.0 \n";
        return Response.ok(msg).build();
    }
}
```

Die Anwendung implementiert einen einfachen REST-Service, der den Hostnamen des Servers zurückgibt, auf dem sie ausgeführt wird.

► 2. Erstellen Sie ein neues Projekt.

- 2.1. Führen Sie mit dem Befehl „source“ die Konfiguration Ihrer Kursumgebung aus:

```
[student@workstation D0288-apps]$ source /usr/local/etc/ocp4.config
```

- 2.2. Melden Sie sich bei OpenShift als Entwicklerbenutzer an:

```
[student@workstation D0288-apps]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
```

- 2.3. Erstellen Sie ein neues Projekt zum Hosten der Anwendung:

```
[student@workstation D0288-apps]$ oc new-project \
${RHT_OCP4_DEV_USER}-manage-builds
```

► 3. Erstellen Sie eine neue Anwendung.

Kapitel 4 | Verwalten von Builds auf OpenShift

- 3.1. Erstellen Sie eine neue Anwendung anhand von Quellen in Git. Verwenden Sie die folgenden Parameter für den Build:

- Anwendungsname: `jhost`
- Branch: `manage-builds`
- Build-Umgebungsvariable: `MAVEN_MIRROR_URL=http://${RHT_OCP4_NEXUS_SERVER}/repository/java`
- Image-Stream: `redhat-openjdk18-openshift:1.8`
- Build-Verzeichnis: `java-serverhost`

Sie können das Skript `/home/student/D0288/labs/manage-builds/oc-new-app.sh` oder den Befehl manuell ausführen:

```
[student@workstation D0288-apps]$ oc new-app --name jhost \
--build-env MAVEN_MIRROR_URL=http://${RHT_OCP4_NEXUS_SERVER}/repository/java \
-i redhat-openjdk18-openshift:1.8 \
https://github.com/${RHT_OCP4_GITHUB_USER}/D0288-apps#manage-builds \
--context-dir java-serverhost
...output omitted...
imagestream.image.openshift.io "jhost" created
buildconfig.build.openshift.io "jhost" created
deployment.apps "jhost" created
service "jhost" created
...output omitted...
```

**Anmerkung**

Verwenden Sie nach der Umgebungsvariable `MAVEN_MIRROR_URL` kein Leerzeichen. Die Umgebungsvariable folgt dem Format `NAME=VALUE`.

- 3.2. Verwenden Sie den Befehl `oc logs`, um die Build-Logs aus dem Build `jhost` zu prüfen:

```
[student@workstation D0288-apps]$ oc logs -f bc/jhost
...output omitted...
Writing manifest to image destination
Storing signatures
...output omitted...
Push successful
```

- 3.3. Warten Sie, bis die Anwendung bereit ist und ausgeführt wird:

```
[student@workstation D0288-apps]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
jhost-1-build  0/1     Completed  0          32m
jhost-7d9b748448-qwtqs 1/1     Running   0          30m
```

- 3.4. Stellen Sie die Anwendung für den externen Zugriff bereit:

```
[student@workstation D0288-apps]$ oc expose svc/jhost
route.route.openshift.io/jhost exposed
```

- 3.5. Rufen Sie den Hostnamen für die neue Route ab:

```
[student@workstation D0288-apps]$ oc get route
NAME      HOST/PORT
jhost     jhost-youruser-manage-builds.apps.cluster.domain.example.com ...
```

- 3.6. Testen Sie die Anwendung:

```
[student@workstation D0288-apps]$ curl \
http://jhost-${RHT_OCP4_DEV_USER}-manage-builds.${RHT_OCP4_WILDCARD_DOMAIN}
I am running on server jhost-1-10mbp Version 1.0
```



Anmerkung

Wenn der vorherige Befehl eine HTML-Seite mit dem Text **Application is not available** zurückgibt, warten Sie einige Sekunden, und versuchen Sie es erneut.

- 4. Listen Sie die Build-Konfigurationen und Builds auf.

- 4.1. Listen Sie alle Build-Konfigurationen im Projekt auf:

```
[student@workstation D0288-apps]$ oc get bc
NAME      TYPE      FROM          LATEST
jhost     Source    Git@manage-builds  1
```

Der Befehl `oc new-app` hat die Build-Konfigurationsressource `jhost` mit der `Source`-Strategie und einer `Git`-Eingabequelle erstellt.

- 4.2. Über die mit dem Befehl `oc new-app` erstellte Build-Konfiguration wurde ein Build gestartet. Listen Sie alle im Projekt verfügbaren Builds auf:

```
[student@workstation D0288-apps]$ oc get builds
NAME      TYPE      FROM          STATUS      STARTED           DURATION
jhost-1   Source    Git@cb73a3d  Complete   18 minutes ago  2m4s
```

- 5. Aktualisieren Sie die Anwendung auf Version 2.0.

- 5.1. Bearbeiten Sie die Datei `/home/student/D0288-apps/java-serverhost/src/main/java/com/redhat/training/example/javaserverhost/rest/ServerHostEndPoint.java`, und aktualisieren Sie auf Version 2.0:

```
... code omitted ...
String msg = "I am running on server "+host+" Version `2.0 \n";
return Response.ok(msg).build();
... code omitted ...
```

- 5.2. Übergeben Sie die Änderungen an den Git-Server.

Kapitel 4 | Verwalten von Builds auf OpenShift

```
[student@workstation D0288-apps]$ cd java-serverhost
[student@workstation java-serverhost]$ git commit -a -m 'Update the version'
...output omitted...
1 file changed, 1 insertion(+), 1 deletion(-)
[student@workstation java-serverhost]$ git push
[student@workstation java-serverhost]$ cd ..
```

5.3. Starten Sie einen neuen Build mit dem Befehl `oc start-build`:

```
[student@workstation D0288-apps]$ oc start-build bc/jhost
build.build.openshift.io/jhost-2 started
```

5.4. Die Anwendung enthält nicht die Änderungen, die Sie in Ihrem lokalen Git-Repository übernommen haben. Die Anwendung verwendet die Remote-Git-Repository-URL als Quelle. Sie müssen die Änderungen an das Remote-Git-Repository übertragen, bevor Sie den OpenShift-Build starten.

Brechen Sie den Build ab, um zu vermeiden, dass bei einer neuen Bereitstellung die ältere Version verwendet wird:

```
[student@workstation D0288-apps]$ oc cancel-build bc/jhost
build.build.openshift.io/jhost-2 marked for cancellation, waiting to be cancelled
build.build.openshift.io/jhost-2 cancelled
```

5.5. Überprüfen Sie, ob der Build abgebrochen wurde:

```
[student@workstation D0288-apps]$ oc get builds
NAME      TYPE      FROM      STATUS      ...
jhost-1   Source    Git@cb73a3d Complete   ...
jhost-2   Source    Git@cb73a3d Cancelled (CancelledBuild) ...
```

Wenn Sie den Build vor dem Abschluss nicht abgebrochen haben, wird folgende Ausgabe ausgegeben:

```
[student@workstation D0288-apps]$ oc get builds
NAME      TYPE      FROM      STATUS      ...
jhost-1   Source    Git@cb73a3d Complete   ...
jhost-2   Source    Git@20d7733  Complete   ...
```

Fahren Sie mit den folgenden Schritten fort.

5.6. Übertragen Sie den aktualisierten Quellcode an den Git-Server:

```
[student@workstation D0288-apps]$ git push
...output omitted...
2aa4b3a..f70977b manage-builds -> manage-builds
[student@workstation java-serverhost]$ cd
```

5.7. Starten Sie einen neuen Build, um die aktualisierte Version der Anwendung bereitzustellen:

```
[student@workstation ~]$ oc start-build bc/jhost
build.build.openshift.io/jhost-3 started
```

5.8. Listen Sie alle Builds auf:

```
[student@workstation ~]$ oc get builds
NAME      TYPE      FROM      STATUS      ...
jhost-1   Source    Git@cb73a3d Complete   ...
jhost-2   Source    Git@cb73a3d Cancelled (CancelledBuild) ...
jhost-3   Source    Git        Running    ...
```

Überprüfen Sie, ob drei Builds verfügbar sind. Der erste Build wurde mit dem Befehl `oc new-app` erstellt, der zweite Build wurde von Ihnen abgebrochen und der dritte Build stellt die aktualisierte Anwendung dar.

5.9. Verfolgen Sie die Build-Logs der Anwendung:

```
[student@workstation ~]$ oc logs -f build/jhost-3
...output omitted...
Push successful
```

5.10. Warten Sie, bis die Anwendung bereit ist und ausgeführt wird:

```
[student@workstation ~]$ oc get pods
NAME          READY  STATUS    RESTARTS  AGE
jhost-1-build  0/1    Completed  0          53m
jhost-5978d79042-kqrwd 1/1    Running   0          4m40s
jhost-3-build  0/1    Completed  0          6m21s
```

5.11. Testen Sie die aktualisierte Anwendung:

```
[student@workstation ~]$ curl \
http://jhost-${RHT_OCP4_DEV_USER}-manage-builds.${RHT_OCP4_WILDCARD_DOMAIN}
I am running on server jhost-2-10mbp Version 2.0
```



Anmerkung

Wenn der vorherige Befehl eine HTML-Seite mit dem Text `Application is not available` zurückgibt, warten Sie einige Sekunden, und versuchen Sie es erneut.

► 6. Bereinigen und löschen Sie das Projekt:

```
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-manage-builds
project.project.openshift.io "youruser-manage-builds" deleted
```

Beenden

Führen Sie auf `workstation` das Skript `lab manage-builds finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht

auf zukünftige Übungen auswirken. Die Aktion „Beenden“ veröffentlicht dieses Projekt und seine Ressourcen.

```
[student@workstation ~]$ lab manage-builds finish
```

Hiermit ist die angeleitete Übung beendet.

Auslösen von Builds

Ziele

Nach Abschluss dieses Abschnitts sollten Sie in der Lage sein, den Build-Prozess mit unterstützten Methoden auszulösen.

Definieren von Build-Triggern

In Red Hat OpenShift können Sie Build-Trigger definieren, damit die Plattform automatisch neue Builds basierend auf bestimmten Ereignissen starten kann. Sie können mit diesen Build-Triggern Ihre Anwendungscontainer mit neuen Container-Images oder Codeänderungen, die sich auf Ihre Anwendung auswirken, auf dem neuesten Stand halten. Red Hat OpenShift definiert zwei Arten von Build-Triggern:

Trigger für Image-Änderungen

Trigger für Image-Änderungen bewirken die Neuerstellung von Container-Images für Anwendungen, um Änderungen zu implementieren, die vom entsprechenden übergeordneten Image vorgenommen wurden.

Webhook-Trigger

Red Hat OpenShift-Webhook-Trigger sind HTTP-API-Endpunkte, die einen neuen Build starten. Verwenden Sie einen Webhook, um Red Hat OpenShift in Ihr Versionskontrollsystem (VCS) wie GitHub oder BitBucket zu integrieren und neue Builds für Codeänderungen auszulösen.

Trigger für Image-Änderungen übernehmen die Aufgabe des Entwicklers zur Überwachung von Änderungen im übergeordneten Image einer Anwendung. Trigger für Image-Änderungen können automatisch von der internen Red Hat OpenShift-Registry aktiviert werden, wenn ein neues Image erkannt wird. Wenn das Image aus einer externen Registry stammt, müssen Sie den Befehl `oc import-image` regelmäßig ausführen, um zu überprüfen, ob sich das Container-Image auf dem Registry-Server geändert hat, damit Sie auf dem neuesten Stand bleiben.

Bei Ausführung des Befehls `oc new-app` werden unter Verwendung der Source- bzw. Docker-Strategie automatisch Trigger für Image-Änderungen für Anwendungen erstellt:

- Bei der Source-Strategie entspricht das übergeordnete Image dem S2I-Builder-Image für die Programmiersprache der Anwendung.
- Bei der Docker-Strategie entspricht das übergeordnete Image dem Image, auf das über die `FROM`-Anweisung im Containerfile der Anwendung verwiesen wird.

Um die einer Build-Konfiguration zugeordneten Trigger anzuzeigen, verwenden Sie den Befehl `oc describe bc`, wie im folgenden Beispiel gezeigt:

```
[user@host ~]$ oc describe bc/name
```

Führen Sie den Befehl `oc set triggers` aus, um einer Build-Konfiguration einen Trigger für Image-Änderungen hinzuzufügen:

```
[user@host ~]$ oc set triggers bc/name --from-image=project/image:tag
```

Kapitel 4 | Verwalten von Builds auf OpenShift

Eine einzelne Build-Konfiguration kann nicht mehrere Trigger für Image-Änderungen enthalten.

Führen Sie den Befehl `oc set triggers` mit der Option `--remove` aus, um einen Trigger für Image-Änderungen aus einer Build-Konfiguration zu entfernen:

```
[user@host ~]$ oc set triggers bc/name --from-image=project/image:tag --remove
```

Führen Sie den Befehl `oc set triggers --help` aus, um die Optionen anzuzeigen, die zum Hinzufügen und Entfernen eines Triggers für Konfigurationsänderungen verwendet wurden.

Starten von Builds mit Webhook-Triggern

Red Hat OpenShift-Webhook-Trigger sind HTTP-API-Endpunkte, die neue Builds starten.

Verwenden Sie einen Webhook, um ein Versionskontrollsystem (VCS) wie Git so zu integrieren, dass Änderungen am Quellcode der Anwendung einen neuen Build mit dem neuesten Code in Red Hat OpenShift auslösen.

Für die Verwendung dieser API-Endpunkte können auch andere Softwarelösungen als Versionskontrollsysteme eingesetzt werden, Red Hat OpenShift-Builds können jedoch Quellcode ausschließlich von einem Git-Server abrufen.

Red Hat OpenShift Container Platform bietet spezielle Webhooks, die API-Endpunkte unterstützen, die mit den folgenden VCS-Services kompatibel sind:

- GitLab
- GitHub
- Bitbucket

Red Hat OpenShift Container Platform bietet auch generische Webhooks, die von Red Hat OpenShift definierte Payload nutzen. Generische Webhooks können von jeder beliebigen Software zum Starten eines Red Hat OpenShift-Builds verwendet werden. Informationen zur Syntax der generischen Webhook-Payload und zu den HTTP-API-Anforderungen für die verschiedenen Webhooks finden Sie in den Referenzen für die Produktdokumentation am Ende dieses Abschnitts.

Für alle Webhooks müssen Sie ein Secret mit einem Schlüssel namens `WebHookSecretKey` und dem Wert definieren, der beim Aufrufen des Webhook angegeben werden soll. Die Webhook-Definition muss dann auf das Secret verweisen. Das Secret stellt die Eindeutigkeit der URL sicher und verhindert, dass andere den Build auslösen. Der Wert des Schlüssels wird mit dem Secret verglichen, das während des Webhook-Aufrufs bereitgestellt wird. Jedes Mal, wenn Sie einen Trigger erstellen oder Red Hat OpenShift automatisch einen erstellt, wird standardmäßig auch ein Secret erstellt.

Mit dem Befehl `oc new-app` werden ein generischer und ein Git-Webhook erstellt. Führen Sie den Befehl `oc set triggers` aus, um einer Build-Konfiguration weitere Arten von Webhooks hinzuzufügen. Gehen Sie beispielsweise folgendermaßen vor, um einer Build-Konfiguration einen GitLab-Webhook hinzuzufügen:

```
[user@host ~]$ oc set triggers bc/name --from-gitlab
```

Wenn die Build-Konfiguration bereits einen GitLab-Webhook enthält, wird durch den vorherigen Befehl das `AuthentifizierungsSecret` in der URL zurückgesetzt. Damit Sie die neue Webhook-URL verwenden können, müssen Sie Ihre GitLab-Projekte aktualisieren.

Führen Sie den Befehl `oc set triggers` mit der Option `--remove` aus, um einen vorhandenen Webhook aus einer Build-Konfiguration zu entfernen: Gehen Sie beispielsweise folgendermaßen vor, um einen GitLab-Webhook aus einer Build-Konfiguration zu entfernen:

```
[user@host ~]$ oc set triggers bc/name --from-gitlab --remove
```

Der Befehl `oc set triggers` unterstützt auch `--from-github-` und `--from-bitbucket-`-Optionen zum Erstellen von Triggern, die für jede VCS-Plattform spezifisch sind.

Um eine Webhook-URL und das Secret abzurufen, führen Sie den Befehl `oc describe` aus und suchen den benötigten, spezifischen Webhook.



Literaturhinweise

Weitere Informationen über Build-Trigger finden Sie im Kapitel *Triggering and Modifying Builds* des Handbuchs *Builds* für Red Hat OpenShift Container Platform 4.6 unter

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/builds/triggering-builds-build-hooks

► Angeleitete Übung

Auslösen von Builds

In dieser Übung lösen Sie einen neuen Build einer Anwendung in Red Hat OpenShift aus, um Aktualisierungen zu implementieren, die am S2I-Builder-Image der Anwendung vorgenommen wurden.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Erstellen einer Anwendung, die auf dem S2I-Builder-Image basiert
- Übertragen einer neuen Version des S2I-Builder-Images
- Aktualisieren der Metadaten in einem Image-Stream als Reaktion auf die-Image-Aktualisierung
- Überprüfen, ob die Anwendung für die Verwendung des neuen S2I-Builder-Images neu erstellt wird

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen ausgeführten Red Hat OpenShift-Cluster
- Auf die Originalversion des PHP-S2I-Builder-Images (`php-73-ubi8-original.tar.gz`)
- Auf die neue Version des PHP-S2I-Builder-Images (`php-73-ubi8-newer.tar.gz`)
- Auf die Beispielanwendung im Git-Repository (`trigger-builds`)

Führen Sie den folgenden Befehl auf der `workstation`-VM aus, um die Voraussetzungen zu überprüfen und die Dateien herunterzuladen, die zum Durchführen dieser Übung erforderlich sind:

```
[student@workstation ~]$ lab trigger-builds start
```

Anweisungen

► 1. Bereiten Sie die Umgebung für die praktische Übung vor.

1.1. Führen Sie mit dem Befehl „`source`“ die Konfiguration der Kursumgebung aus:

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

1.2. Melden Sie sich bei Red Hat OpenShift mit Ihrem Entwicklerbenutzernamen an:

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
```

- 1.3. Erstellen Sie ein neues Projekt für die Anwendung:

```
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-trigger-builds
```

- 2. Fügen Sie dem Projekt einen Image-Stream hinzu, der mit der neuen Anwendung verwendet werden soll.

- 2.1. Melden Sie sich mit Podman bei Ihrem persönlichen Quay.io-Benutzerkonto an.

```
[student@workstation ~]$ podman login -u ${RHT_OCP4_QUAY_USER} quay.io  
Password:  
Login Succeeded!
```

- 2.2. Übertragen Sie das ursprüngliche PHP 7.3-Builder-Image an Ihre Quay.io-Registry.

Sie können den Befehl mit dem Skript /home/student/D0288/labs/trigger-builds/push-original.sh kopieren oder ausführen:

```
[student@workstation ~]$ cd /home/student/D0288/labs/trigger-builds  
[student@workstation trigger-builds]$ skopeo copy \  
oci-archive:php-73-ubi8-original.tar.gz \  
docker://quay.io/${RHT_OCP4_QUAY_USER}/php-73-ubi8:latest  
Getting image source signatures  
...output omitted...  
Writing manifest to image destination  
Storing signatures
```

- 2.3. Die Quay.io-Registry ist standardmäßig für private Images vorgesehen, sodass Sie dem Builder-Servicekonto ein Secret hinzufügen müssen, um darauf zugreifen zu können.

Erstellen Sie ein Secret aus dem Zugriffstoken der Container-Registry-API, das von Podman gespeichert wurde.

```
[student@workstation trigger-builds]$ oc create secret generic quay-registry \  
--from-file .dockerconfigjson=${XDG_RUNTIME_DIR}/containers/auth.json \  
--type kubernetes.io/dockerconfigjson  
secret/quay-registry created
```

Fügen Sie dem Builder-Servicekonto das Quay.io-Registry-Secret hinzu.

```
[student@workstation trigger-builds]$ oc secrets link builder quay-registry
```

- 2.4. Aktualisieren Sie den php-Image-Stream, um die Metadaten für das neue Container-Image abzurufen. Die externe Registry verwendet das docker-distribution-Paket und benachrichtigt Red Hat OpenShift nicht über Image-Änderungen.

```
[student@workstation trigger-builds]$ oc import-image php \  
--from quay.io/${RHT_OCP4_QUAY_USER}/php-73-ubi8 --confirm  
imagestream.image.openshift.io/php-73-ubi8 imported
```

```
Name:          php  
...output omitted...
```

Kapitel 4 | Verwalten von Builds auf OpenShift

```
latest
  tagged from quay.io/youruser/php-73-ubi8

* quay.io/youruser/php-73-ubi8@sha256:c919...8cb2
  Less than a second ago
...output omitted...
```

► 3. Erstellen Sie eine neue Anwendung.

- 3.1. Erstellen Sie einen neuen Anwendungs-Build mit den folgenden Parametern:

- Name: **trigger**
- Builder Image: **php**
- Verzeichnis der Anwendung: **trigger-builds**

Sie können den vollständigen Befehl kopieren oder ausführen, indem Sie das Skript /home/student/D0288/labs/trigger-builds/oc-new-app.sh nutzen:

```
[student@workstation trigger-builds]$ oc new-app \
--name trigger \
php~http://github.com/${RHT_OCP4_GITHUB_USER}/D0288-apps \
--context-dir trigger-builds
--> Found image ... „youruser-trigger-builds/php“ ... for „php“

...output omitted...

--> Success
Build scheduled, use 'oc logs -f buildconfig/trigger' to track its progress.
...output omitted...
```

- 3.2. Warten Sie, bis der Build abgeschlossen ist.

```
[student@workstation trigger-builds]$ oc logs -f bc/trigger
...output omitted...
Push successful
```

- 3.3. Warten Sie, bis die Anwendung bereit ist und ausgeführt wird:

```
[student@workstation trigger-builds]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
trigger-1-build 0/1     Completed  0          10m
trigger-1-q6bln 1/1     Running   0          9m2s
```

- 3.4. Überprüfen Sie, ob ein Trigger für Image-Änderungen als Bestandteil der Build-Konfiguration definiert ist:

```
[student@workstation trigger-builds]$ oc describe bc/trigger | grep Triggered
Triggered by: Config, ImageChange
```

Mit dem letzten Trigger **ImageChange** wird ein neuer Build gestartet, wenn der Image-Stream feststellt, dass das zugehörige Basis-Image geändert wurde.

- 4. Aktualisieren Sie den Image-Stream, um einen neuen Build zu starten.
- 4.1. Laden Sie die neue Version des PHP-S2I-Builder-Images in die Quay.io-Registry hoch.
Sie können den vollständigen Befehl kopieren oder ausführen, indem Sie das Skript /home/student/D0288/labs/trigger-builds/push-newer.sh nutzen:
- ```
[student@workstation trigger-builds]$ skopeo copy \
oci-archive:php-73-ubi8-newer.tar.gz \
docker://quay.io/${RHT_OCP4_QUAY_USER}/php-73-ubi8:latest
Getting image source signatures
...output omitted...
Writing manifest to image destination
Storing signatures
```
- 4.2. Aktualisieren Sie den php-Image-Stream, um die Metadaten für das neue Container-Image abzurufen.
- ```
[student@workstation trigger-builds]$ oc import-image php
imagestream.image.openshift.io/php-73-ubi8 imported

Name:          php
...output omitted...
latest
tagged from quay.io/youruser/php-73-ubi8

* quay.io/youruser/php-73-ubi8@sha256:3f5e...6ab7
  Less than a second ago
  quay.io/youruser/php-73-ubi8@sha256:c919...8cb2
    2 minutes ago
...output omitted...
```
- 5. Überprüfen Sie den neuen Build.
- 5.1. Durch die Aktualisierung des Image-Streams wurde ein neuer Build ausgelöst. Listen Sie alle Builds auf und überprüfen Sie, ob ein zweiter Build gestartet wurde:
- ```
[student@workstation trigger-builds]$ oc get builds
NAME TYPE FROM STATUS STARTED DURATION
trigger-1 Source Git@da010df Complete 13 minutes ago 58s
trigger-2 Source Git@da010df Complete 28 seconds ago 15s
```
- 5.2. Überprüfen Sie, ob der trigger-2-Build durch die Aktualisierung eines Image-Streams gestartet wurde:
- ```
[student@workstation trigger-builds]$ oc describe build trigger-2 | grep cause
Build trigger cause: Image change
```
- 6. Beenden.
- 6.1. Wechseln Sie in das Benutzerverzeichnis.

Kapitel 4 | Verwalten von Builds auf OpenShift

```
[student@workstation trigger-builds]$ cd
```

- 6.2. Löschen Sie das Container-Image aus der externen Registry:

```
[student@workstation ~]$ skopeo delete \
docker://quay.io/${RHT_OCP4_QUAY_USER}/php-73-ubi8
```

- 6.3. Melden Sie sich bei Quay.io mit Ihrem persönlichen kostenlosen Benutzerkonto an.
6.4. Klicken Sie im Menü der obersten Ebene von Quay.io auf **Repositories**, und filtern Sie Repositorys mit dem Namen **php-73-ubi8**. Klicken Sie dann auf das Repository **php-73-ubi8**.

The screenshot shows a list of repositories on Quay.io. At the top, there's a search bar with the text 'php-73-ubi8'. Below it, a table lists one repository: 'youruser/php-73-ubi8'. The repository details show it was last modified 'Yesterday at 2:01 PM'. There are buttons for 'ACTIVITY' and 'STAR' on the right. A red box highlights the search bar and the repository name in the table.

- 6.5. Klicken Sie auf der Seite **Repository Activity** für das Repository **php-73-ubi8** auf das Zahnradssymbol. Scrollen Sie nach unten bis zur Schaltfläche **Delete Repository** und klicken Sie darauf.

The screenshot shows the 'Activity' page for the 'php-73-ubi8' repository. It includes sections for 'Repository Visibility' (private) and 'Delete Repository'. A warning message says 'Deleting a Repository cannot be undone. Here be dragons!' next to a red 'Delete Repository' button, which is highlighted with a red box.

- 6.6. Klicken Sie im Fenster **Delete** auf **Delete**, um zu bestätigen, dass Sie das Repository **php-73-ubi8** löschen möchten.

The screenshot shows a modal dialog titled 'Delete'. It contains a warning message 'This action cannot be undone!' and a question 'Continue with deletion of this Repository?'. At the bottom are 'Delete' and 'Cancel' buttons. A red box highlights the 'Delete' button. Below the dialog, a red banner repeats the warning message 'Deleting a Repository cannot be undone. Here be dragons!' and the 'Delete Repository' button.

Beenden

Führen Sie auf `workstation` das Skript `lab trigger-builds finish` aus, um diese Übung abzuschließen. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken. Die Aktion „Beenden“ veröffentlicht dieses Projekt und seine Ressourcen.

```
[student@workstation ~]$ lab trigger-builds finish
```

Hiermit ist die angeleitete Übung beendet.

Implementieren des Build-Hook „post-commit“

Ziele

Nach Abschluss dieses Abschnitts sollten Sie in der Lage sein, die Logik im Anschluss an die Erstellung mithilfe des Build-Hook „post-commit“ zu bearbeiten.

Beschreiben der „post-commit“-Build-Hooks

Red Hat OpenShift Container Platform (RHOC) stellt die Funktionalität für das post-commit-Build Hook bereit, die Validierungen während der Erstellung ermöglicht. Mithilfe des Build-Hook post-commit werden Befehle in einem temporären Container ausgeführt, bevor das neue Container-Image, das vom Build generiert wurde, zur Registry übertragen wird. Der Build-Hook startet mit dem vom Build erstellten Container-Image einen temporären Container.

Abhängig vom Beendigungscode der Befehlsausführung im temporären Container wird das Image von RHOC an die Registry übertragen oder nicht. Wenn der Befehl einen Beendigungscode ungleich 0 zurückgibt, der einen Fehlschlag angibt, überträgt RHOC das Image nicht und kennzeichnet den Build als fehlgeschlagen. Wenn der Befehl erfolgreich ausgeführt wird, überträgt RHOC das Image an die Registry.

Sie können überprüfen, ob ein Build aufgrund eines „post-commit“-Hook fehlgeschlagen ist, indem Sie die Build-Logs mit dem Befehl `oc logs` untersuchen:

```
[user@host ~]$ oc logs bc/name
...
Running post commit hook ...
...
error: build error: container "openshift_s2i-build_hook-2_post-commit_post-
commit_45ec0816" returned non-zero exit code: 35
```



Anmerkung

Post-Commit-Hooks dienen nur zur Validierung und niemals zur Änderung eines Image.

Überprüfen von Anwendungsfällen für „post-commit“-Build-Hooks

Ein typisches Szenario für die Verwendung eines „post-commit“-Build-Hook besteht darin, Tests in Ihrer Anwendung auszuführen. Anhand eines Tests kann vor der Übertragung des Images durch RHOC an die Registry und der Initiierung einer neuen Bereitstellung überprüft werden, ob die Anwendung ordnungsgemäß funktioniert. Wenn der Test fehlschlägt, schlägt der Build fehl und wird nicht mit einer Bereitstellung fortgesetzt.

Es gibt einige andere häufige Anwendungsfälle, in denen es nützlich sein kann, einen „post-commit“-Build-Hook zu nutzen. Beispiel:

- Zur Integration eines Builds in eine externe Anwendung über eine HTTP-API

- Zur Überprüfung einer nicht funktionalen Anforderung wie Anwendungsleistung, Sicherheit, Benutzerfreundlichkeit oder Kompatibilität
- Zum Senden einer E-Mail an das Entwicklerteam mit Informationen über den neuen Build

Konfigurieren eines „post-commit“-Build-Hook

Es gibt zwei Arten von Build-Hooks des Typs „post-commit“, die Sie konfigurieren können:

Befehl

Ein Befehl wird über den Systemaufruf `exec` ausgeführt. Erstellen Sie einen „post-commit“-Build-Hook-Befehl mit der Option `--command`, wie im folgenden Befehl gezeigt:

```
[user@host ~]$ oc set build-hook bc/name --post-commit \
--command -- bundle exec rake test --verbose
```



Anmerkung

Das Leerzeichen direkt nach dem Argument `--` ist kein Fehler. Das Argument `--` trennt die RHOCB-Befehlszeile, in der der „post-commit“-Hook konfiguriert wird, von dem Befehl, der vom Hook ausgeführt wird.

Shell-Skript

Führt einen Build-Hook mit dem Befehl `/bin/sh -ic` aus. Dieses Verfahren ist bequemer, da alle Funktionen – wie die Erweiterung und Umleitung von Argumenten – von einer Shell bereitgestellt werden. Dieses Verfahren ist nur möglich, wenn das Basis-Image die sh-Shell verwendet. Erstellen Sie ein Shell-Skript für den „post-commit“-Build-Hook mit der Option `--script`, wie im folgenden Befehl gezeigt:

```
[user@host ~]$ oc set build-hook bc/name --post-commit \
--script="curl http://api.com/user/${USER}"
```



Literaturhinweise

Weitere Informationen zu „post-commit“-Build-Hooks finden Sie im Kapitel *Triggering and Modifying Builds* des Handbuchs *Builds* für Red Hat OpenShift Container Platform 4.6 unter
<https://access.redhat.com/documentation/en-us/openShiftContainerPlatform/4.6/html-single/builds/triggering-builds-build-hooks>

► Angeleitete Übung

Implementieren des Build-Hook „post-commit“

In dieser Übung legen Sie einen „post-commit“-Build-Hook für die Integration eines Builds in eine externe Anwendung fest.

Ergebnisse

Sie sollten in der Lage sein, einer Build-Konfigurationsressource einen „post-commit“-Build-Hook hinzuzufügen. Der „post-commit“-Build-Hook sendet eine HTTP-API-Anforderung an die Anwendung `builds-for-managers`.

Bevor Sie Beginnen

Die Anwendung `builds-for-managers` wird von Managern zur Nachverfolgung von Anwendungs-Builds verwendet, die von einem Entwicklerteam erstellt wurden. In der Anwendung werden verschiedene Informationen angezeigt, darunter das Projekt, die Git-URL und der Entwickler, der den Build gestartet hat. Sie müssen Ihre Builds in die Anwendung integrieren, damit Manager Ihre Arbeit nachverfolgen können.

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen ausgeführten Red Hat OpenShift-Cluster
- Auf das PHP-S2I-Builder-Image (`php-73-rhel7:7.3`).
- Auf das `builds-for-managers`-Container-Image (`quay.io/redhattraining/builds-for-managers`)
- Auf das Git-Repository der Anwendung (`post-commit`)

Führen Sie den folgenden Befehl auf der `workstation`-VM aus, um die Voraussetzungen zu validieren, das `post-commit`-Projekt zu erstellen, die Anwendung `builds-for-managers` zu starten und die Dateien herunterzuladen, die zum Durchführen dieser Übung erforderlich sind:

```
[student@workstation ~]$ lab post-commit start
```

Anweisungen

► 1. Bereiten Sie die Umgebung für die praktische Übung vor.

- 1.1. Laden Sie die Konfiguration Ihrer Kursumgebung.

Führen Sie den folgenden Befehl aus, um die in der ersten angeleiteten Übung erstellten Umgebungsvariablen zu laden:

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

- 1.2. Melden Sie sich beim Red Hat OpenShift-Cluster an.

```
[student@workstation D0288-apps]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful
...output omitted...
```

- 1.3. Stellen Sie sicher, dass Sie sich im Projekt `youruser-post-commit` befinden:

```
[student@workstation ~]$ oc project ${RHT_OCP4_DEV_USER}-post-commit
Already on project "youruser-post-commit" on server "https://...".
```

Da der Befehl `lab post-commit start` dieses Projekt erstellt, sollten Sie sich bereits in diesem Projekt befinden. Ist dies nicht der Fall, kann sich Ihre Ausgabe von der obigen unterscheiden.

- 1.4. Überprüfen Sie, ob `builds-for-managers` im Projekt `youruser-post-commit` ausgeführt wird:

```
[student@workstation ~]$ oc status
In project youruser-post-commit on server https://api.cluster.domain....
http://builds-for-managers... to pod port 8080-tcp (svc/builds-for-managers)
  deployment/builds-for-managers deploys istag/builds-for-managers:latest
    deployment #1 deployed 5 minutes ago - 1 pod
...output omitted...
```

► 2. Erstellen Sie eine neue Anwendung.

- 2.1. Erstellen Sie eine neue Anwendung anhand von Quellen in Git. Geben Sie der Anwendung den Namen `hook` und stellen Sie der Git-Repository-URL den Image-Stream `php:7.3` mit einer Tilde (~) voran.

Der vollständige Befehl kann entweder kopiert oder direkt über das Skript `oc-new-app.sh` im Ordner `/home/student/D0288/labs/post-commit` ausgeführt werden:

```
[student@workstation ~]$ oc new-app --name hook --context-dir post-commit \
php:7.3~http://github.com/${RHT_OCP4_GITHUB_USER}/D0288-apps
```

- 2.2. Warten Sie, bis der Build abgeschlossen ist:

```
[student@workstation ~]$ oc logs -f bc/hook
...output omitted...
Push successful
```

- 2.3. Warten Sie, bis die Anwendung bereit ist und ausgeführt wird:

```
[student@workstation ~]$ oc get pods
NAME                  READY   STATUS    RESTARTS   AGE
builds-for-managers-1-w1s8f   1/1     Running   0          8m
hook-1-build           0/1     Completed  0          1m
hook-1-lmn12           1/1     Running   0          11s
```

Es werden derzeit zwei verschiedene Anwendungen ausgeführt. Die erste ist die Anwendung `builds-for-managers`, die von Managern verwendet wird, und die zweite ist Ihre PHP-Anwendung.

► 3. Integrieren Sie den PHP-Anwendungs-Build in die Anwendung `builds-for-managers`.

- 3.1. Überprüfen Sie das Skript `create-hook.sh`, das im Ordner `/home/student/D0288/labs/post-commit` bereitgestellt wurde. Das Skript erstellt einen Build-Hook, über den Ihre PHP-Anwendungs-Builds mit dem Befehl `curl` in die Anwendung `builds-for-managers` integriert werden.

```
[student@workstation ~]$ cat ~/D0288/labs/post-commit/create-hook.sh
...output omitted...
oc set build-hook bc/hook --post-commit --command -- \
  bash -c "curl -s -S -i -X POST http://builds-for-managers-
${RHT_OCP4_DEV_USER}-post-commit.${RHT_OCP4_WILDCARD_DOMAIN}/api/builds
-f -d 'developer=\${DEVELOPER}&git=\${OPENSOURCE_BUILD_SOURCE}&project=\
\${OPENSOURCE_BUILD_NAMESPACE}'"
```

Mit dem Befehl `curl` werden drei Umgebungsvariablen an die Anwendung `builds-for-managers` gesendet:

- `DEVELOPER`: Enthält den Namen des Entwicklers.
- `OPENSOURCE_BUILD_SOURCE`: Enthält die Git-URL des Projekts.
- `OPENSOURCE_BUILD_NAMESPACE`: Enthält den Namen des Projekts.

3.2. Führen Sie das Skript `create-hook.sh` aus:

```
[student@workstation ~]$ ~/D0288/labs/post-commit/create-hook.sh
buildconfig.build.openshift.io/hook hooks updated
```

3.3. Überprüfen Sie, ob der „post-commit“-Build-Hook erstellt wurde:

```
[student@workstation ~]$ oc describe bc/hook | grep Post
Post Commit Hook: ["bash", "-c", "\"curl -s -S -i -X POST http://builds-...
```

- 3.4. Starten Sie mit dem Befehl `oc start-build` und der aktivierten Option `-F` einen neuen Build, um die Logs anzuzeigen und den HTTP-API-Antwortcode zu überprüfen. Der Befehl `curl`, der vom „post-commit“-Hook ausgeführt wird, gibt den Statuscode „HTTP 400“ zurück:

```
[student@workstation ~]$ oc start-build bc/hook -F
build.build.openshift.io/hook-2 started
...output omitted...
STEP 11: RUN bash -c "curl -s -S -i -X POST http://builds-for-managers-...
curl: (22) The requested URL returned error: 400 Bad Request
subprocess exited with status 22
subprocess exited with status 22
error: build error: error building at STEP "RUN bash -c "curl -s -S -i ...
```

Die Anwendung `builds-for-managers` hat die HTTP-API-Anforderung abgelehnt, da die Umgebungsvariable `DEVELOPER` nicht definiert wurde.

- 3.5. Zeigen Sie die Builds an und überprüfen Sie, ob ein Build aufgrund eines Fehlers im „post-commit“-Hook fehlgeschlagen ist:

```
[student@workstation ~]$ oc get builds
NAME      TYPE      FROM      STATUS    ...output omitted...
hook-1    Source    Git@c2166cc Complete
hook-2    Source    Git@c2166cc Failed (GenericBuildFailed)
```

► 4. Beheben Sie das Problem mit der fehlenden Umgebungsvariablen.

- 4.1. Erstellen Sie die Build-Umgebungsvariable `DEVELOPER` unter Angabe Ihres Namens mit dem Befehl `oc set env`:

```
[student@workstation ~]$ oc set env bc/hook DEVELOPER="Your Name"
buildconfig.build.openshift.io/hook updated
```

- 4.2. Überprüfen Sie, ob die Umgebungsvariable `DEVELOPER` in der hook-Build-Konfiguration verfügbar ist:

```
[student@workstation ~]$ oc set env bc/hook --list
# buildconfigs hook
DEVELOPER=Your Name
```

- 4.3. Starten Sie einen neuen Build und überprüfen Sie in den Logs den HTTP-API-Antwortcode. Der Statuscode „HTTP 200“ wird angezeigt:

```
[student@workstation ~]$ oc start-build bc/hook -F
build.build.openshift.io/hook-3 started
...output omitted...
STEP 11: RUN bash -c "curl -s -S -i -X POST http://builds-for-managers-...
HTTP/1.1 200 OK
...output omitted...
Push successful
```

Der Statuscode „HTTP 200“ gibt an, dass die Anwendung `builds-for-managers` die HTTP API-Anfrage zugelassen hat.

- 4.4. Öffnen Sie einen Webbrowser, um auf `http://builds-for-managers-youruser-post-commit.apps.cluster.domain.example.com` zuzugreifen.

```
[student@workstation ~]$ firefox $(oc get route/builds-for-managers \
-o jsonpath='{.spec.host}') &
```

Auf der Seite werden alle Builds sowie der Entwickler angezeigt, der den jeweiligen Build gestartet hat.

Builds for Managers

Date	Developer	Project	Git Project
2019-07-10 07:08:43	Your Name	youruser-post-commit	http://github.com/youruser/DO288-apps

- 5. Führen Sie eine Bereinigung durch. Löschen Sie das Projekt *youruser-post-commit* in Red Hat OpenShift.

```
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-post-commit
```

Beenden

Führen Sie auf *workstation* das Skript *lab post-commit finish* aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken. Die Aktion „Beenden“ veröffentlicht dieses Projekt und seine Ressourcen.

```
[student@workstation ~]$ lab post-commit finish
```

Hiermit ist die angeleitete Übung beendet.

► Praktische Übung

Erstellen von Anwendungen für OpenShift

In dieser praktischen Übung verwenden Sie Red Hat OpenShift zur Verwaltung von Anwendungs-Builds, zur Behebung von Anwendungsproblemen und zur Auslösung eines neuen Builds mithilfe von Webhooks.



Anmerkung

Für das Bewertungsskript am Ende jeder praktischen Übung eines Kapitels ist es erforderlich, dass Sie die exakten Projektnamen und anderen Identifikatoren, wie in der Spezifikation der praktischen Übung angegeben, verwenden.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Beheben von Anwendungsproblemen durch die Verwaltung des Lifecycle eines Builds
- Auslösen eines neuen Builds mithilfe von Webhooks

Bevor Sie Beginnen

Zum Durchführen dieser Aufgabe benötigen Sie Zugriff:

- Auf einen ausgeführten Red Hat OpenShift-Cluster
- Auf das S2I-Builder-Image und den Image-Stream für Node.js-Anwendungen (nodejs)
- Auf die Anwendung im Git-Repository (`build-app`)

Führen Sie den folgenden Befehl auf der `workstation`-VM aus, um die Voraussetzungen zu überprüfen und die Dateien herunterzuladen, die zum Durchführen dieser Übung erforderlich sind:

```
[student@workstation ~]$ lab build-app start
```

Anforderungen

Die bereitgestellte Anwendung ist in JavaScript mit der Node.js-Laufzeit geschrieben. Es handelt sich um eine einfache Anwendung, die auf dem Express-Framework basiert. Sie sollten die Anwendung entsprechend den folgenden Anforderungen erstellen und auf einem Red Hat OpenShift-Cluster bereitstellen:

- Der Projektname lautet `youruser-build-app`.
- Der Anwendungsname lautet `simple`. Verwenden Sie das Skript `oc-new-app.sh` im Verzeichnis der praktischen Übung, um die Anwendung zu erstellen und bereitzustellen. Dieses Skript enthält einen absichtlichen Fehler, den Sie in einem späteren Schritt beheben. Ändern oder bearbeiten Sie das Skript `oc-new-app.sh` nicht.

Kapitel 4 | Verwalten von Builds auf OpenShift

- Die bereitgestellte Anwendung wird aus dem Quellcode im Unterverzeichnis `build-app` des Git-Repository erstellt:

`https://github.com/youruser/D0288-apps.`

- Die zum Erstellen der Anwendung erforderlichen NPM-Module sind über die Nexus-Server-URL verfügbar, unter:

`http://${RHT_OCP4_NEXUS_SERVER}/repository/nodejs.`

Verwenden Sie die Umgebungsvariable `npm_config_registry`, um diese Informationen an das S2I-Builder-Image für Node.js weiterzugeben.

- Der Zugriff auf die Anwendung erfolgt über die Standardroute:

`simple-youruser-build-app.apps.cluster.domain.example.com`.

Anweisungen

- Erstellen Sie das Projekt `youruser-build-app`.
- Führen Sie das Skript `/home/student/D0288/labs/build-app/oc-new-app.sh` aus, um die Anwendung zu erstellen.



Wichtig

Das Skript `oc-new-app.sh` enthält einen absichtlichen Fehler, den Sie in einem späteren Schritt beheben. Ändern oder bearbeiten Sie das Skript `oc-new-app.sh` nicht.

- Überprüfen Sie, ob der Anwendungs-Build fehlschlägt, und beheben Sie das Problem.
- Stellen Sie den Anwendungsservice für den externen Zugriff bereit und rufen Sie die Routen-URL ab.
- Starten Sie einen neuen Build und überprüfen Sie, ob die Anwendung bereit ist und ausgeführt wird. Überprüfen Sie, ob auf die Anwendung mit der im vorherigen Schritt abgerufenen Routen-URL zugegriffen werden kann.
- Verwenden Sie den generischen Webhook für die Build-Konfiguration, um einen neuen Anwendungs-Build zu starten.
- Bewerten Sie Ihre Arbeit:

```
[student@workstation ~]$ lab build-app grade
```

- Bereinigen und löschen Sie das Projekt.

```
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-build-app
```

Beenden

Führen Sie auf `workstation` das Skript `lab build-app finish` aus, um diese Übung abzuschließen. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken. Die Aktion „Beenden“ veröffentlicht dieses Projekt und seine Ressourcen.

```
[student@workstation ~]$ lab build-app finish
```

Hiermit ist die praktische Übung abgeschlossen.

► Lösung

Erstellen von Anwendungen für OpenShift

In dieser praktischen Übung verwenden Sie Red Hat OpenShift zur Verwaltung von Anwendungs-Builds, zur Behebung von Anwendungsproblemen und zur Auslösung eines neuen Builds mithilfe von Webhooks.



Anmerkung

Für das Bewertungsskript am Ende jeder praktischen Übung eines Kapitels ist es erforderlich, dass Sie die exakten Projektnamen und anderen Identifikatoren, wie in der Spezifikation der praktischen Übung angegeben, verwenden.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Beheben von Anwendungsproblemen durch die Verwaltung des Lifecycle eines Builds
- Auslösen eines neuen Builds mithilfe von Webhooks

Bevor Sie Beginnen

Zum Durchführen dieser Aufgabe benötigen Sie Zugriff:

- Auf einen ausgeführten Red Hat OpenShift-Cluster
- Auf das S2I-Builder-Image und den Image-Stream für Node.js-Anwendungen (`nodejs`)
- Auf die Anwendung im Git-Repository (`build-app`)

Führen Sie den folgenden Befehl auf der `workstation`-VM aus, um die Voraussetzungen zu überprüfen und die Dateien herunterzuladen, die zum Durchführen dieser Übung erforderlich sind:

```
[student@workstation ~]$ lab build-app start
```

Anforderungen

Die bereitgestellte Anwendung ist in JavaScript mit der Node.js-Laufzeit geschrieben. Es handelt sich um eine einfache Anwendung, die auf dem Express-Framework basiert. Sie sollten die Anwendung entsprechend den folgenden Anforderungen erstellen und auf einem Red Hat OpenShift-Cluster bereitstellen:

- Der Projektname lautet `youruser-build-app`.
- Der Anwendungsname lautet `simple`. Verwenden Sie das Skript `oc-new-app.sh` im Verzeichnis der praktischen Übung, um die Anwendung zu erstellen und bereitzustellen. Dieses Skript enthält einen absichtlichen Fehler, den Sie in einem späteren Schritt beheben. Ändern oder bearbeiten Sie das Skript `oc-new-app.sh` nicht.

- Die bereitgestellte Anwendung wird aus dem Quellcode im Unterverzeichnis build-app des Git-Repository erstellt:

<https://github.com/youruser/D0288-apps>.

- Die zum Erstellen der Anwendung erforderlichen NPM-Module sind über die Nexus-Server-URL verfügbar, unter:

[http://\\${RHT_OCP4_NEXUS_SERVER}/repository/nodejs](http://${RHT_OCP4_NEXUS_SERVER}/repository/nodejs).

Verwenden Sie die Umgebungsvariable `npm_config_registry`, um diese Informationen an das S2I-Builder-Image für Node.js weiterzugeben.

- Der Zugriff auf die Anwendung erfolgt über die Standardroute:

`simple-youruser-build-app.apps.cluster.domain.example.com`.

Anweisungen

- Erstellen Sie das Projekt `youruser-build-app`.

- Laden Sie die Konfiguration Ihrer Kursumgebung.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

- Melden Sie sich bei Red Hat OpenShift mit Ihrem Entwicklerbenutzerkonto an:

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
```

- Erstellen Sie ein neues Projekt zum Hosten der Anwendung:

```
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-build-app
```

- Führen Sie das Skript `/home/student/D0288/labs/build-app/oc-new-app.sh` aus, um die Anwendung zu erstellen.



Wichtig

Das Skript `oc-new-app.sh` enthält einen absichtlichen Fehler, den Sie in einem späteren Schritt beheben. Ändern oder bearbeiten Sie das Skript `oc-new-app.sh` nicht.

- Überprüfen Sie das Skript, mit dem die Anwendung erstellt wird:

```
[student@workstation ~]$ cat ~/D0288/labs/build-app/oc-new-app.sh
...output omitted...
oc new-app --name simple --build-env \
  npm_config_registry=http://invalid-server:8081/nexus/content/groups/nodejs \
  https://github.com/${RHT_OCP4_GITHUB_USER}/D0288-apps \
  --context-dir build-app
```

Kapitel 4 | Verwalten von Builds auf OpenShift

- 2.2. Führen Sie das Skript `oc-new-app.sh` aus, um die Anwendung zu erstellen:

```
[student@workstation ~]$ ~/DO288/labs/build-app/oc-new-app.sh  
...output omitted...  
--> Creating resources ...  
...output omitted...  
--> Success  
...output omitted...
```

3. Überprüfen Sie, ob der Anwendungs-Build fehlschlägt, und beheben Sie das Problem.

- 3.1. Zeigen Sie die Build-Logs an, um den Build-Fehler zu identifizieren. Es kann einige Zeit dauern, bis eine Fehlermeldung angezeigt wird.

```
[student@workstation ~]$ oc logs -f bc/simple  
...output omitted...  
error: build error: error building at STEP "RUN /usr/libexec/s2i/assemble": error  
while running runtime: exit status 1  
...output omitted...
```

- 3.2. Eine ungültige Nexus-Server-URL hat den Fehler verursacht. Überprüfen Sie, ob die Nexus-Server-URL fehlerhaft ist:

```
[student@workstation ~]$ oc set env bc simple --list  
# buildconfigs simple  
npm_config_registry=http://invalid-server:8081/nexus/content/groups/nodejs
```

- 3.3. Korrigieren Sie die Variable, damit die korrekte Nexus-Server-URL verwendet wird:

```
[student@workstation ~]$ oc set env bc simple \  
npm_config_registry=http://${RHT_OCP4_NEXUS_SERVER}/repository/nodejs  
buildconfig.build.openshift.io/simple updated
```

Beachten Sie, dass vor und hinter dem Gleichheitszeichen (=) nach `npm_config_registry` kein Leerzeichen steht.

- 3.4. Überprüfen Sie, ob die Variable `npm_config_registry` den richtigen Wert aufweist.

```
[student@workstation ~]$ oc set env bc simple --list  
# buildconfigs simple  
npm_config_registry=http://nexus-common.../repository/nodejs
```

Beachten Sie, dass vor und hinter dem Gleichheitszeichen (=) nach `npm_config_registry` kein Leerzeichen steht. Das komplette „Schlüssel=Wert“-Paar für die Build-Umgebungsvariable ist zu lang für die Papierbreite.

4. Stellen Sie den Anwendungsservice für den externen Zugriff bereit und rufen Sie die Routen-URL ab.

- 4.1. Stellen Sie den Service bereit:

```
[student@workstation ~]$ oc expose svc simple  
route.route.openshift.io/simple exposed
```

- 4.2. Rufen Sie die Routen-URL ab:

```
[student@workstation ~]$ oc get route/simple \
-o jsonpath='{.spec.host}{"\n"}'
simple-youruser-build-app.apps.cluster.domain.example.com
```

5. Starten Sie einen neuen Build und überprüfen Sie, ob die Anwendung bereit ist und ausgeführt wird. Überprüfen Sie, ob auf die Anwendung mit der im vorherigen Schritt abgerufenen Routen-URL zugegriffen werden kann.

- 5.1. Starten Sie einen neuen Build und beobachten Sie die Logs:

```
[student@workstation ~]$ oc start-build simple
build.build.openshift.io/simple-2 started
...output omitted...
Push successful
```

- 5.2. Warten Sie, bis die Anwendung bereit ist und ausgeführt wird:

NAME	READY	STATUS	RESTARTS	AGE
simple-1-build	0/1	Error	0	20m
simple-2-build	0/1	Completed	0	4m5s
simple-964487d7b-fs9gr	1/1	Running	0	5m41s

- 5.3. Testen Sie die Anwendung:

```
[student@workstation ~]$ curl \
simple-${RHT_OCP4_DEV_USER}-build-app.${RHT_OCP4_WILDCARD_DOMAIN}
Simple app for the Building Applications Lab!
```

6. Verwenden Sie den generischen Webhook für die Build-Konfiguration, um einen neuen Anwendungs-Build zu starten.

- 6.1. Rufen Sie mit dem Befehl `oc describe bc` die generische Webhook-URL ab, mit der ein neuer Build gestartet wird.

```
[student@workstation ~]$ oc describe bc simple
Name: simple
...output omitted...
Webhook Generic:
URL: https://apps.cluster.domain.example.com/apis/build.openshift.io/v1/
namespaces/youruser-build-app/buildconfigs/simple/webhooks/<secret>/generic
```

- 6.2. Rufen Sie mit dem Befehl `oc get bc` das Secret für den Webhook ab und übergeben Sie die Option `-o json`, um die Details der Build-Konfiguration in JSON auszugeben.

```
[student@workstation ~]$ SECRET=$(oc get bc simple \
-o jsonpath='{.spec.triggers[*].generic.secret}{"\n"}')
```

Kapitel 4 | Verwalten von Builds auf OpenShift

- 6.3. Starten Sie einen neuen Build mit der Webhook-URL und dem Secret, die im vorherigen Schritt in der Ausgabe enthalten waren. Die Fehlermeldung über den „ungültigen Inhaltstyp für Payload“ kann ohne Weiteres ignoriert werden.

```
[student@workstation ~]$ curl -X POST -k \
${RHT_OCP4_MASTER_API}\
/apis/build.openshift.io/v1/namespaces/${RHT_OCP4_DEV_USER}-build-app\
/buildconfigs/simple/webhooks/$SECRET/generic
...output omitted...
"status": "Success",
...output omitted...
```

**Anmerkung**

Der vorhergehende Befehl enthält nach der ersten Zeile keine Leerzeichen. Wenn der Befehl „curl“ fehlschlägt, überprüfen Sie, ob:

- Ihre URL Leerzeichen enthält.
- die Variable RHT_OCP4_MASTER_API mit einem Schrägstrich (/) endet. Ist dies der Fall, verwenden Sie \${RHT_OCP4_MASTER_API%}/ im vorhergehenden Befehl, um den Schrägstrich zu entfernen.
- Ihre \$SECRET-Variable das richtige Secret enthält.

- 6.4. Listen Sie alle Builds auf und überprüfen Sie, ob ein neuer Build gestartet wurde:

```
[student@workstation ~]$ oc get builds
NAME      TYPE      FROM      STATUS      STARTED ...
simple-1  Source    Git@3e14daf  Failed (AssembleFailed)  About an hour ago
simple-2  Source    Git@3e14daf  Complete    20 minutes ago
simple-3  Source    Git@3e14daf  Complete    32 seconds ago
```

- 6.5. Warten Sie, bis der neue Build abgeschlossen wird:

```
[student@workstation ~]$ oc logs -f bc/simple
...output omitted...
Push successful
```

- 6.6. Warten Sie, bis die Anwendung bereit ist und ausgeführt wird:

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
simple-1-build 0/1     Error     0          94m
simple-2-build 0/1     Completed  0          36m
simple-3-build 0/1     Completed  0          19m
simple-7c6995cdcf-phvk5 1/1     Running   0          16m
```

7. Bewerten Sie Ihre Arbeit:

```
[student@workstation ~]$ lab build-app grade
```

8. Bereinigen und löschen Sie das Projekt.

```
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-build-app
```

Beenden

Führen Sie auf `workstation` das Skript `lab build-app finish` aus, um diese Übung abzuschließen. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken. Die Aktion „Beenden“ veröffentlicht dieses Projekt und seine Ressourcen.

```
[student@workstation ~]$ lab build-app finish
```

Hiermit ist die praktische Übung abgeschlossen.

Zusammenfassung

In diesem Kapitel wurden die folgenden Themen behandelt:

- Eine `BuildConfig`-Ressource umfasst eine Strategie und eine oder mehrere Eingabequellen.
- Es gibt vier Build-Strategien: `Source`, `Pipeline`, `Docker` und `Custom`.
- Die sechs Build-Eingabequellen nach Rangfolge sind: „`Dockerfile`“, „`Git`“, „`image`“, „`binary`“, „`input secrets`“ und „`external artifacts`“.
- Verwalten Sie den Build-Lifecycle mit `oc`-CLI-Befehlen wie `oc start-build`, `oc cancel-build`, `oc delete`, `oc describe` und `oc logs`.
- Builds können automatisch mit Build-Triggern gestartet werden, z. B. mit Triggern zur Änderung von Images und Webhooks.
- Sie können während Builds mit dem Build-Hook `post-commit` Validierungen durchführen.

Kapitel 5

Anpassen von Source-to-Image-Builds

Ziel

Anpassen eines vorhandenen S2I-Builder-Images und Erstellen eines neuen Builder-Images

Ziele

- Beschreiben der obligatorischen und optionalen Schritte beim Source-to-Image-Build-Prozess
- Anpassen eines vorhandener S2I-Builder-Images mit Skripts
- Erstellen eines neuen S2I-Builder-Images mit S2I-Tools

Abschnitte

- Beschreiben der Source-to-Image-Architektur (und Test)
- Anpassen eines vorhandenen S2I-Builder-Images (und angeleitete Übung)
- Erstellen eines S2I-Base-Images (und angeleitete Übung)

Praktische Übung

Anpassen von Source-to-Image-Builds

Beschreiben der Source-to-Image-Architektur

Ziele

Nach Abschluss dieses Abschnitts sollten Sie in der Lage sein, die obligatorischen und optionalen Schritte in einem Source-to-Image-Build-Prozess zu beschreiben.

Source-to-Image-Spracherkennung (S2I)

Red Hat OpenShift (RHOCP) kann Anwendungen direkt aus dem Quellcode erstellen, der in einem Git-Repository gespeichert ist. Bei der einfacheren Syntax für den Befehl `oc new-app` ist nur die Git-Repository-URL als Argument erforderlich. Red Hat OpenShift versucht anschließend, die von der Anwendung verwendete Programmiersprache automatisch zu erkennen und wählt ein kompatibles Builder-Image aus.

Die Logik für die automatische Erkennung ist nicht vollständig ausgereift. Mit dem Befehl `oc new-app` können verschiedene Befehlszeilenoptionen verwendet werden, um eine spezifische Auswahlmöglichkeit zu erzwingen. Diese Befehlszeilenoptionen wurden in diesem Kurs bereits erläutert.

Die Erkennungsfunktion für Programmiersprachen basiert auf der Suche nach spezifischen Dateinamen im root-Verzeichnis des Git-Repository. Die folgende Tabelle enthält einige der gebräuchlicheren Optionen, es handelt sich jedoch nicht um eine umfangreiche Liste aller mit Source-to-Image kompatiblen Sprachen. In der Produktdokumentation zur jeweiligen Version von RHOCP finden Sie alle Regeln:

Ressourcen	Sprachen-Builder	Programmiersprache
Dockerfile	n. a.	Dockerfile-Build (nicht S2I)
pom.xml	jee	Java (mit JBoss EAP)
app.json, package.json	nodejs	Node.js (JavaScript)
composer.json, index.php	php	PHP

RHOCP folgt einem mehrstufigen Algorithmus, um zu bestimmen, ob die URL auf ein Quellcode-Repository verweist. Falls dies zutrifft, legt RHOCP fest, mit welchem Builder-Image der Build durchgeführt werden soll. Nachfolgend finden Sie eine vereinfachte Beschreibung des Algorithmus:

1. RHOCP greift auf die URL als Container-Registry-URL zu. Wenn der Zugriff möglich ist, muss keine Build-Konfiguration erstellt werden. RHOCP erstellt die Bereitstellung und andere Ressourcen, die für die Bereitstellung eines Container-Images erforderlich sind.
2. Wenn die URL auf ein Git Repository verweist, ruft RHOCP eine Datei aus dem Repository ab und sucht nach einer Dockerfile-Datei. Falls eine Datei gefunden wird, verwendet die Build-Konfiguration die docker-Strategie. Andernfalls verwendet die Build-Konfiguration die source-Strategie, die ein S2I-Builder-Image erfordert.

3. RHOCP sucht nach Image-Streams, die mit dem Namen des Sprachen-Builders übereinstimmen, der als Wert der `supports`-Annotation angegeben ist. Die erste Übereinstimmung stammt vom S2I-Builder-Image.
4. Wenn keine Annotation übereinstimmt, sucht RHOCP nach einem Image-Stream, dessen Name mit dem Namen des Sprachen-Builders übereinstimmt. Die erste Übereinstimmung stammt vom S2I-Builder-Image.

Die Schritte 3 und 4 vereinfachen den Vorgang zum Hinzufügen neuer Builder-Images zu einem RHOCP-Cluster. Das bedeutet, dass potenziell mehrere Builder-Images übereinstimmen können. In diesem Kapitel wurden die `oc new-app`-Befehlszeilenoptionen bereits erläutert, um diese Mehrdeutigkeit zu vermeiden und um sicherzustellen, dass RHOCP den korrekten Image-Stream als S2I-Builder-Image auswählt.

Wenn Sie beispielsweise den folgenden Befehl ausführen, erkennt der Befehl `oc`, dass Sie auf eine Registry-URL verweisen, und startet die Containerbereitstellung:

```
[user@host ~]$ oc new-app registry.access.redhat.com/ubi8/ubi:8.0
```

Wenn Sie den folgenden Befehl ausführen, erkennt der Befehl `oc` zudem, dass Sie ein Git-Repository verwenden. Der Befehl klont dann das Repository, um nach einer `Dockerfile`-Datei zu suchen. Wenn der RHOCP-Cluster eine `Dockerfile`-Datei im root-Verzeichnis des Repository findet, löst er einen neuen Container-Build-Prozess aus.

```
[user@host ~]$ oc new-app https://github.com/RedHatTraining/D0288-apps/ubi-echo
```

Wenn Ihr RHOCP-Cluster eine der in der vorherigen Tabelle erwähnten Dateien im root-Verzeichnis des Repository findet, startet der RHOCP-Cluster einen S2I-Prozess mit seinem jeweiligen Image-Builder.

Um die Verwendung eines bestimmten Image-Streams zu erzwingen, können Sie die Option `-i` für eine PHP 7.3-Anwendung angeben:

```
[user@host ~]$ oc new-app -i php:7.3 \
https://github.com/RedHatTraining/D0288-apps/php-helloworld
```

Der RHOCP-Cluster sucht nach einem Image-Stream namens `php` und nach der Version 7.3, um den Builder aufzurufen.

Der S2I-Build-Prozess (Source-to-Image)

Der S2I-Build-Prozess umfasst drei grundlegende Komponenten, die zur Erstellung des endgültigen Container-Images miteinander kombiniert werden:

Anwendungsquellcode

Dies ist der Quellcode für die Anwendung.

S2I-Skripts

S2I-Skripts sind Skripts, die der RHOCP-Build-Prozess ausführt, um das S2I-Builder-Image anzupassen. S2I-Skripts können in jeder beliebigen Programmiersprache geschrieben werden, solange die Skripts im S2I-Builder-Image ausgeführt werden können.

Das S2I-Builder-Image

Dies ist ein Container-Image, das die erforderliche Laufzeitumgebung für die Anwendung enthält. Es enthält in der Regel Compiler, Interpreter, Skripts und andere Abhängigkeiten, die zur Ausführung der Anwendung erforderlich sind.

Der S2I-Build-Prozess basiert auf S2I-Skripts, die er in verschiedenen Phasen (Stages) des Build-Workflows ausführt. Die Skripts und eine kurze Beschreibung ihrer Funktionsweise finden Sie in der nachfolgenden Tabelle:

Skript	Obligatorisch?	Beschreibung
assemble	Ja	Mit dem Skript <code>assemble</code> wird die Anwendung über den Quellcode erstellt und im Image in die entsprechenden Verzeichnisse platziert.
run	Ja	Mit dem Skript <code>run</code> wird Ihre Anwendung ausgeführt. Es wird empfohlen, beim Ausführen von Containerprozessen den Befehl <code>exec</code> in Ihrem <code>run</code> -Skript zu verwenden. Dies stellt die Signalweitergabe und das ordnungsgemäße Herunterfahren eines vom Skript <code>run</code> gestarteten Prozesses sicher.
save-artifacts	Nein	Mit dem <code>save-artifacts</code> -Skript werden alle für die Anwendung erforderlichen Abhängigkeiten erfasst und zur Beschleunigung nachfolgender Builds gespeichert. Beispiel: durch Bundler installierte Gems für Ruby oder .m2-Inhalte für Java. Dies bedeutet, dass der Build diese Inhalte nicht erneut herunterladen muss, wodurch die Build-Dauer verringert wird. Diese Abhängigkeiten werden in einer tar-Datei erfasst und an die Standardausgabe gestreamt.
usage	Nein	Mit dem Skript <code>usage</code> wird eine Beschreibung zur ordnungsgemäßen Verwendung Ihres Images bereitgestellt.
test/run	Nein	Mit dem Skript <code>test/run</code> können Sie einen einfachen Prozess erstellen, um zu überprüfen, ob das Image ordnungsgemäß funktioniert.

Der S2I-Build-Workflow

Der S2I-Build-Prozess wird wie folgt durchgeführt:

1. RHOCP instanziert einen Container, der auf dem S2I-Builder-Image basiert, und erstellt dann eine tar-Datei, die den Quellcode und die S2I-Skripts enthält. RHOCP streamt anschließend die tar-Datei in den Container.
2. Vor Ausführung des Skripts `assemble` extrahiert RHOCP die tar-Datei aus dem vorherigen Schritt und speichert die Inhalte am Speicherort, der entweder durch die Option `--destination` oder die Bezeichnung `io.openshift.s2i.destination` aus dem Builder-Image angegeben wird. Der Standardspeicherort ist das Verzeichnis `/tmp`.
3. Wenn es sich hierbei um einen inkrementellen Build handelt, stellt das Skript `assemble` die Build-Artefakte wieder her, die zuvor vom Skript `save-artifacts` in einer tar-Datei gespeichert wurden.

4. Mit dem Skript `assemble` wird die Anwendung aus dem Quellcode erstellt und die Binärdateien werden in die entsprechenden Verzeichnisse platziert.
5. Wenn es sich hierbei um einen inkrementellen Build handelt, wird das Skript `save-artifacts` ausgeführt und alle abhängigen Build-Artefakte werden in einer tar-Datei gespeichert.
6. Nach Abschluss des `assemble`-Skripts wird der Container committet, um das endgültige Image zu erstellen. RHOCP legt das `run`-Skript als CMD-Anweisung für das endgültige Image fest.

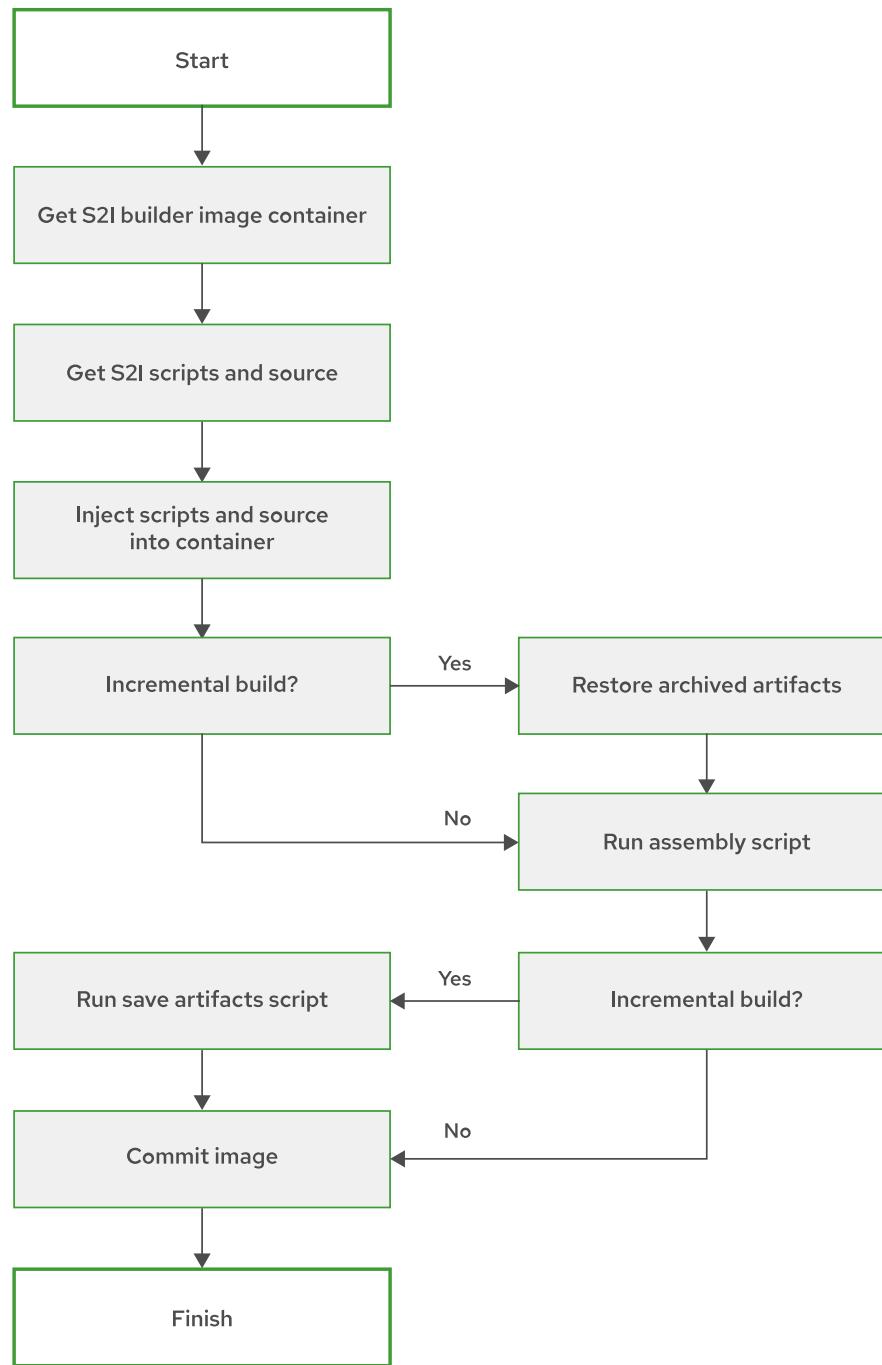


Abbildung 5.1: Der S2I-Build-Workflow

Kapitel 5 | Anpassen von Source-to-Image-Builds

Um ein Builder-Image zu erstellen, deklarieren Sie eine **Dockerfile**-Datei mit den Tools, die zum Erstellen der Anwendung erforderlich sind, z. B. Compiler, Build-Tools und alle zuvor genannten Skriptdateien. Die folgende **Dockerfile**-Builder-Datei definiert einen NGINX-Server-Builder:

```
FROM registry.access.redhat.com/ubi8/ubi:8.0

LABEL io.k8s.description="My custom Builder" \
      io.k8s.display-name="Nginx 1.6.3" \
      io.openshift.expose-services="8080:http" \
      io.openshift.tags="builder,webserver,html,nginx" \
      io.openshift.s2i.scripts-url="image:///usr/libexec/s2i" ②

RUN yum install -y epel-release && \
    yum install -y --nодocs nginx && \
    yum clean all

EXPOSE 8080 ④

COPY ./s2i/bin/ /usr/libexec/s2i ⑤
```

- ①** Legen Sie die Bezeichnungen fest, die für RHOPC verwendet werden, um das Builder-Image zu beschreiben.
- ②** Teilen Sie S2I mit, wo sich die obligatorischen Skripts (`run`, `assemble`) befinden.
- ③** Installieren Sie das NGINX-Webserverpaket und bereinigen Sie den Yum-Cache.
- ④** Legen Sie den Standardport für Anwendungen fest, die mit diesem Image erstellt wurden.
- ⑤** Kopieren Sie die S2I-Skripts in das Verzeichnis `/usr/libexec/s2i`.

Das Assemble-Skript kann wie folgt definiert werden:

```
#!/bin/bash -e

if [ "$(ls -A /tmp/src)" ]; then
    mv /tmp/src/* /usr/share/nginx/html/①
fi
```

- ①** Überschreiben Sie die NGINX-Standarddatei `index.html`.

```
#!/bin/bash -e
/usr/sbin/nginx -g "daemon off;" ①
```

- ①** Verhindern Sie, dass der NGINX-Prozess als Daemon ausgeführt wird, damit der Container nach der Ausführung des Skripts `exec` nicht beendet wird.

Überschreiben von S2I-Builder-Skripts

S2I-Builder-Images stellen S2I-Standardskripts bereit. Sie können die S2I-Standardskripts überschreiben, um die Vorgehensweise zur Erstellung und Ausführung Ihrer Anwendung zu ändern. Sie haben die Möglichkeit, das Standardverhalten des Builds zu überschreiben, ohne ein neues

S2I-Builder-Image zu erstellen, indem Sie einen Fork des Quellcodes für den ursprünglichen S2I-Builder erstellen.

Die S2I-Standardskripts für eine Anwendung lassen sich am einfachsten überschreiben, indem Sie Ihre S2I-Skripts zum Quellcode-Repository für Ihre Anwendung hinzufügen. Sie können im Ordner `.s2i/bin` des Quellcode-Repository Ihrer Anwendung S2I-Skripts bereitstellen.

Wenn RHOPC den S2I-Prozess startet, werden der Quellcodeordner, die benutzerdefinierten S2I-Skripts und der Anwendungsquellcode überprüft. RHOPC fügt alle diese Dateien in die tar-Datei ein, die in das S2I-Builder-Image eingefügt wird. RHOPC führt anschließend das benutzerdefinierte Skript `assemble` anstelle des im S2I-Builder enthaltenen Standardskripts `assemble` aus, gefolgt von den anderen benutzerdefinierten Skripts (sofern vorhanden).



Literaturhinweise

Vorgehensweise zum Überschreiben von S2I-Builder-Skripts

<https://blog.openshift.com/override-s2i-builder-scripts/>

Erstellen eines S2I Builder Image

<https://blog.openshift.com/create-s2i-builder-image/>

Source-to-Image (S2I) Tool

<https://github.com/openshift/source-to-image>

S2I-Befehlszeilenschnittstelle

<https://github.com/openshift/source-to-image/blob/master/docs/cli.md>

Weitere Informationen über die Build-Umgebungsvariablen aus den standardmäßigen S2I-Builder-Images von RHOPC finden Sie im Handbuch *Images* für RHOPC 4.6 unter

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/images/

► Quiz

Beschreiben der Source-to-Image-Architektur

Wählen Sie die richtigen Antworten auf die folgenden Fragen aus:

- ▶ 1. Welche der folgenden S2I-Skripts sind für die Erstellung der binären Anwendungsdateien in einem S2I-Build zuständig?
 - a. run
 - b. assemble
 - c. save-artifacts
 - d. test/run

- ▶ 2. Welche zwei der folgenden Aussagen über den S2I-Build-Prozess sind richtig? (Wählen Sie zwei Antworten aus.)
 - a. Das Skript assemble wird ausgeführt, bevor die tar-Datei mit dem Anwendungsquellcode und den S2I-Skripts extrahiert wird.
 - b. Das Skript assemble wird ausgeführt, nachdem die tar-Datei mit dem Anwendungsquellcode und den S2I-Skripts extrahiert wurde.
 - c. Das Skript run wird ausgeführt, nachdem die tar-Datei mit dem Anwendungsquellcode und den S2I-Skripts extrahiert wurde.
 - d. Das Skript run wird als CMD-Anweisung für das endgültige Container-Image festgelegt.
 - e. Das Skript assemble wird als CMD-Anweisung für das endgültige Container-Image festgelegt.

- ▶ 3. In welchem der folgenden Verzeichnisse (bezogen auf das Root Ihres Quellcodes) würden Sie eigene benutzerdefinierte S2I-Skripts bereitstellen?
 - a. .scripts/bin
 - b. .s2i/bin
 - c. etc/bin
 - d. usr/local/bin

- ▶ 4. Welche zwei der folgenden Skripts sind in einem S2I-Build obligatorisch? (Wählen Sie zwei Antworten aus.)
 - a. usage
 - b. test/run
 - c. assemble
 - d. run
 - e. save-artifacts

► **5. Welche zwei der folgenden Szenarien eignen sich für inkrementelle S2I-Builds? (Wählen Sie zwei Antworten aus.)**

- a. Eine Java EE-Anwendung mit vielen JAR-Abhängigkeiten, die mit Apache Maven verwaltet werden.
- b. Eine Anwendung, die von einer Dumpdatei einer SQL-Datenbank abhängt, die beim Anwendungsstart abgerufen wird.
- c. Eine Ruby-Webanwendung, die über viele statische Ressourcen verfügt, wie Images, CSS- und HTML-Dateien.
- d. Eine Node.js-Anwendung mit Abhängigkeiten, die mithilfe von npm verwaltet werden.

► **6. Welche der folgenden Bezeichnungen geben für das S2I-Builder-Image das Verzeichnis an, in dem die Skripts gespeichert sind?**

- a. io.openshift.s2i.scripts-url
- b. io.openshift.s2i.scripts-dir
- c. io.openshift.s2i.scripts-directory
- d. io.openshift.s2i.scripts-URL

► Lösung

Beschreiben der Source-to-Image-Architektur

Wählen Sie die richtigen Antworten auf die folgenden Fragen aus:

- ▶ 1. Welche der folgenden S2I-Skripts sind für die Erstellung der binären Anwendungsdateien in einem S2I-Build zuständig?
 - a. run
 - b. assemble
 - c. save-artifacts
 - d. test/run

- ▶ 2. Welche zwei der folgenden Aussagen über den S2I-Build-Prozess sind richtig? (Wählen Sie zwei Antworten aus.)
 - a. Das Skript assemble wird ausgeführt, bevor die tar-Datei mit dem Anwendungsquellcode und den S2I-Skripts extrahiert wird.
 - b. Das Skript assemble wird ausgeführt, nachdem die tar-Datei mit dem Anwendungsquellcode und den S2I-Skripts extrahiert wurde.
 - c. Das Skript run wird ausgeführt, nachdem die tar-Datei mit dem Anwendungsquellcode und den S2I-Skripts extrahiert wurde.
 - d. Das Skript run wird als CMD-Anweisung für das endgültige Container-Image festgelegt.
 - e. Das Skript assemble wird als CMD-Anweisung für das endgültige Container-Image festgelegt.

- ▶ 3. In welchem der folgenden Verzeichnisse (bezogen auf das Root Ihres Quellcodes) würden Sie eigene benutzerdefinierte S2I-Skripts bereitstellen?
 - a. .scripts/bin
 - b. .s2i/bin
 - c. etc/bin
 - d. usr/local/bin

- ▶ 4. Welche zwei der folgenden Skripts sind in einem S2I-Build obligatorisch? (Wählen Sie zwei Antworten aus.)
 - a. usage
 - b. test/run
 - c. assemble
 - d. run
 - e. save-artifacts

► **5. Welche zwei der folgenden Szenarien eignen sich für inkrementelle S2I-Builds? (Wählen Sie zwei Antworten aus.)**

- a. Eine Java EE-Anwendung mit vielen JAR-Abhängigkeiten, die mit Apache Maven verwaltet werden.
- b. Eine Anwendung, die von einer Dumpdatei einer SQL-Datenbank abhängt, die beim Anwendungsstart abgerufen wird.
- c. Eine Ruby-Webanwendung, die über viele statische Ressourcen verfügt, wie Images, CSS- und HTML-Dateien.
- d. Eine Node.js-Anwendung mit Abhängigkeiten, die mithilfe von npm verwaltet werden.

► **6. Welche der folgenden Bezeichnungen geben für das S2I-Builder-Image das Verzeichnis an, in dem die Skripts gespeichert sind?**

- a. io.openshift.s2i.scripts-url
- b. io.openshift.s2i.scripts-dir
- c. io.openshift.s2i.scripts-directory
- d. io.openshift.s2i.scripts-URL

Anpassen eines vorhandenen S2I-Builder-Images

Ziele

Nach Abschluss dieses Abschnitts sollten Sie in der Lage sein, die S2I-Skripts eines vorhandenen S2I-Builder-Images anzupassen.

Anpassen von Skripts eines vorhandenen S2I-Builder-Images

Die S2I-Skripts werden standardmäßig in S2I-Builder-Images gepackt. In bestimmten Szenarien kann es erforderlich sein, diese Skripts anzupassen, um die Vorgehensweise zur Erstellung und Ausführung Ihrer Anwendung zu ändern, ohne das Image neu erstellen zu müssen.

Der S2I-Build-Prozess bietet eine Methode zum Überschreiben der S2I-Standardskripts. Sie können im Ordner `.s2i/bin` des Anwendungsquellcodes eigene S2I-Skripts bereitstellen. Während des Build-Prozesses werden die benutzerdefinierten S2I-Skripts automatisch erkannt und anstelle der S2I-Standardskripts im Builder-Image ausgeführt.

Abhängig vom Umfang der Anpassung, die an den überschriebenen Skripts vorgenommen werden muss, können Sie die Standard-S2I-Skripts vollständig durch eine eigene Version ersetzen. Alternativ können Sie ein *Wrapper*-Skript erstellen, das die Standardskripts abruft und anschließend die Änderungen vor bzw. nach dem Abruf hinzufügt.

Beispiel: Nehmen wir an, Sie möchten die S2I-Skripts für das S2I-Builder-Image `rhscl/php-73-rhel7` anpassen und die Vorgehensweise zur Erstellung und Ausführung der Anwendung ändern. Sie können das folgende Verfahren verwenden, um die S2I-Skripts in diesem Builder-Image anzupassen:

- Rufen Sie mit dem Befehl `podman pull` das Container-Image aus einer Container-Registry auf das lokale System ab. Ermitteln Sie mit dem Befehl `podman inspect` den Wert der Bezeichnung `io.openshift.s2i.scripts-url`, um den Standardspeicherort der S2I-Skripts im Image zu ermitteln.

```
[user@host ~]$ podman pull \
myregistry.example.com/rhscl/php-73-rhel7
...output omitted...
Digest: sha256:...
[user@host ~]$ podman inspect \
--format='{{ index .Config.Labels "io.openshift.s2i.scripts-url"}}' \
rhscl/php-73-rhel7
image:///usr/libexec/s2i
```

- Sie haben auch die Möglichkeit, den Befehl `skopeo inspect` auszuführen, um dieselben Informationen direkt aus einer externen Registry abzurufen:

```
[user@host ~]$ skopeo inspect \
docker://myregistry.example.com/rhscl/php-73-rhel7 \
| grep io.openshift.s2i.scripts-url
"io.openshift.s2i.scripts-url": "image:///usr/libexec/s2i",
```

- Erstellen Sie im Ordner .s2i/bin einen Wrapper für das Skript assemble:

```
#!/bin/bash
echo "Making pre-invocation changes..."

/usr/libexec/s2i/assemble
rc=$?

if [ $rc -eq 0 ]; then
    echo "Making post-invocation changes..."
else
    echo "Error: assemble failed!"
fi

exit $rc
```

- Erstellen Sie ebenso im Ordner .s2i/bin einen Wrapper für das Skript run:

```
#!/bin/bash
echo "Before calling run..."
exec /usr/libexec/s2i/run
```



Anmerkung

Beim Erstellen eines Wrapper für das Skript run müssen Sie es mit exec aufrufen. Dadurch wird sichergestellt, dass das Standardskript run weiterhin mit der Prozess-ID 1 ausgeführt wird. Andernfalls kann es während des Herunterfahrens zu Fehlern bei der Signalweitergabe und zu einer fehlerhaften Ausführung Ihrer Anwendung kommen. Das bedeutet auch, dass Sie nach Abrufen des Standardskripts run keine weiteren Befehle ausführen können.

Inkrementelle Builds in S2I

Bei der Erstellung von Anwendungen auf einem Red Hat OpenShift-Cluster (RHOC) mittels S2I werden üblicherweise kleine, inkrementelle Änderungen für Ihre Anwendungen erstellt, bereitgestellt und getestet. Einige Organisationen setzen Verfahren zur *kontinuierlichen Integration (Continuous Integration, CI)* und *kontinuierlichen Bereitstellung (Continuous Delivery, CD)* ein, bei denen die Anwendung ohne manuelle Eingriffe mehrmals in schnellen, sich wiederholenden Zyklen erstellt und bereitgestellt wird.

Wenn Ihre Anwendung modular mit mehreren abhängigen Komponenten und Libraries erstellt wird, nehmen S2I-Builds aufgrund der „unveränderlichen Natur“ der Container viel Zeit in Anspruch. Während des Build-Prozesses müssen die Abhängigkeiten abgerufen werden. Anschließend muss die Anwendung bei jeder Änderung am Quellcode neu erstellt und bereitgestellt werden.

Kapitel 5 | Anpassen von Source-to-Image-Builds

Der S2I-Build-Prozess bietet einen Mechanismus zur Reduzierung des Zeitaufwands bei der Erstellung von Builds, indem das Skript `save-artifacts` im Rahmen des S2I-Lifecycle nach dem Skript `assemble` abgerufen wird. Das Skript `save-artifacts` stellt sicher, dass abhängige Artefakte (Libraries und Komponenten, die für die Anwendung erforderlich sind) für künftige Builds gespeichert werden.

Während des nächsten Build-Prozesses stellt das Skript `assemble` die zwischengespeicherten Artefakte wieder her, bevor die Anwendung aus dem Quellcode erstellt wird. Beachten Sie, dass das Skript `save-artifacts` für das Streamen von Abhängigkeiten zur Standardausgabe in einer tar-Datei zuständig ist.



Wichtig

Die Ausgabe des Skripts `save-artifacts` darf nur die Stream-Ausgabe der tar-Datei enthalten. Die Ausgabe anderer Befehle im Skript sollte zur Datei `/dev/null` umgeleitet werden.

Beispiel: Nehmen wir an, Sie entwickeln eine Java EE-basierte Anwendung mit vielen Abhängigkeiten, die mithilfe von Apache Maven verwaltet werden. Nehmen Sie weiter an, Sie haben ein S2I-Builder-Image erstellt, bei dem die JAR-Datei der Anwendung durch das Skript `assemble` kompiliert und gepackt wird: In diesem Fall reduzieren inkrementelle Builds, die die zuvor heruntergeladenen JAR-Dateien wiederverwenden können, den Zeitaufwand für die Erstellung von Builds erheblich. Apache Maven speichert JAR-Abhängigkeiten im Ordner `$HOME/.m2`.

Das Skript `save-artifacts`, das die Maven-JAR-Dateien zwischenspeichert, kann wie folgt definiert werden:

```
#!/bin/sh -e

# Stream the .m2 folder tar archive to stdout
if [ -d ${HOME}/.m2 ]; then
    pushd ${HOME} > /dev/null
    tar cf - .m2
    popd > /dev/null
fi
```

Der entsprechende Code zur Wiederherstellung der Artefakte vor der Erstellung im Skript `assemble`:

```
# Restore the .m2 folder
...output omitted...
if [ -d /tmp/artifacts/.m2 ]; then
    echo "---> Restoring maven artifacts..."
    mv /tmp/artifacts/.m2 ${HOME}/
fi
...output omitted...
```



Literaturhinweise

Weitere Informationen finden Sie im Kapitel *Creating Images* des Handbuchs *Images* für RHOC 4.6 unter
[https://access.redhat.com/documentation/en-us/
openShift/container_platform/4.6/html-single/images/index#creating-images](https://access.redhat.com/documentation/en-us/openShift/container_platform/4.6/html-single/images/index#creating-images)

Vorgehensweise zum Überschreiben von S2I-Builder-Skripts

<https://blog.openshift.com/override-s2i-builder-scripts>

► Angeleitete Übung

Anpassen von S2I-Builds

In dieser Übung passen Sie die S2I-Skripts eines vorhandenen S2I-Builder-Images an, um eine Informationsseite zur Anwendung hinzuzufügen.

Ergebnisse

Sie sollten in der Lage sein, die Skripts `assemble` und `run` eines Builder-Images für den Apache HTTP-Server anzupassen. Sie überschreiben die integrierten Standardskripts mit eigenen, benutzerdefinierten Versionen.

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen ausgeführten Red Hat OpenShift-Cluster (RHOCP)
- Auf das S2I-Builder-Image für den Apache HTTP-Server `rhscl/httpd-24-rhel7`
- Auf einen Fork im Git-Repository, der den Anwendungsquellcode `s2i-scripts` enthält

Führen Sie den folgenden Befehl auf der `workstation`-VM aus, um die Voraussetzungen für die Übung zu überprüfen sowie die Übungs- und Lösungsdateien herunterzuladen:

```
[student@workstation ~]$ lab s2i-scripts start
```

Anweisungen

- 1. Identifizieren Sie die S2I-Skripts, die im Builder-Image `rhscl/httpd-24-rhel7` gepackt wurden.
- 1.1. Führen Sie auf der `workstation`-VM das Image `rhscl/httpd-24-rhel7` in einem Terminalfenster aus und überschreiben Sie den Einstiegspunkt des Containers, um eine Shell auszuführen:

```
[student@workstation ~]$ podman run --name test -it rhscl/httpd-24-rhel7 bash
Trying to pull registry.access.redhat.com/rhscl/httpd-24-rhel7:latest...
Getting image source signatures
...output omitted...
bash-4.2$
```

- 1.2. Überprüfen Sie die S2I-Skripts, die im Builder-Image gepackt wurden. Die S2I-Skripts befinden sich im Ordner `/usr/libexec/s2i`:

```
bash-4.2$ cat /usr/libexec/s2i/assemble  
...output omitted...  
bash-4.2$ cat /usr/libexec/s2i/run  
...output omitted...  
bash-4.2$ cat /usr/libexec/s2i/usage  
...output omitted...
```

Beenden Sie den Container:

```
bash-4.2$ exit
```

► 2. Überprüfen Sie den Anwendungsquellcode anhand der benutzerdefinierten S2I-Skripts.

- 2.1. Wechseln Sie zu Ihrem lokalen Klon des D0288-apps-Git-Repository und checken Sie den main-Branch des Kurs-Repository aus, um sicherzustellen, dass Sie diese Übung von einem als fehlerfrei bekannten Zustand aus starten:

```
[student@workstation ~]$ cd D0288-apps  
[student@workstation D0288-apps]$ git checkout main  
...output omitted...
```

- 2.2. Überprüfen Sie die Datei /home/student/D0288-apps/s2i-scripts/index.html.

Die HTML-Datei enthält eine einfache Meldung:

```
Hello Class! D0288 rocks!!!
```

- 2.3. Die benutzerdefinierten S2I-Skripts befinden sich im Ordner /home/student/D0288-apps/s2i-scripts/.s2i/bin. Das Skript .s2i/bin/assemble kopiert die Datei index.html aus der Anwendungsquelle in das Dokumentstammverzeichnis des Webservers unter /opt/app-root/src. Es erstellt zudem eine info.html-Datei mit der Erstellungszeit für die Seite und Informationen zur Umgebung:

```
...output omitted...  
CUSTOMIZATION STARTS HERE  
  
echo "---> Installing application source"  
cp -Rf /tmp/src/*.* ./  
  
DATE=date "+%b %d, %Y @ %H:%M %p"  
  
echo "---> Creating info page"  
echo "Page built on $DATE" >> ./info.html  
echo "Proudly served by Apache HTTP Server version $HTTPD_VERSION" >> ./info.html  
  
CUSTOMIZATION ENDS HERE  
...output omitted...
```

- 2.4. Das Skript .s2i/bin/run ändert den Standard-Log-Level von Startmeldungen im Webserver in debug:

```
# Make Apache show 'debug' level logs during startup
exec run-httpd -e debug $@
```

- 3. Stellen Sie die Anwendung in einem RHOC-P-Cluster bereit. Überprüfen Sie, ob die benutzerdefinierten S2I-Skripts ausgeführt werden.

- 3.1. Laden Sie die Konfiguration Ihrer Kursumgebung.

Führen Sie den folgenden Befehl aus, um die in der ersten angeleiteten Übung erstellten Umgebungsvariablen zu laden:

```
[student@workstation D0288-apps]$ source /usr/local/etc/ocp4.config
```

- 3.2. Melden Sie sich bei RHOC-P mit Ihrem Entwicklerbenutzerkonto an:

```
[student@workstation D0288-apps]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

- 3.3. Erstellen Sie ein neues Projekt für die Anwendung. Stellen Sie dem Projektnamen Ihren Entwicklerbenutzernamen voran.

```
[student@workstation D0288-apps]$ oc new-project ${RHT_OCP4_DEV_USER}-s2i-scripts
Now using project "youruser-s2i-scripts" on server
"https://api.cluster.domain.example.com:6443".
...output omitted...
```

- 3.4. Erstellen Sie eine neue Anwendung mit dem Namen `bonjour` anhand von Quellen in Git. Sie müssen der Git-URL mit der Tilde-Notation den `httpd:2.4`-Image-Stream als Präfix voranstellen, um sicherzustellen, dass die Anwendung das Builder-Image `rhscl/httpd24-rhel7` verwendet.

```
[student@workstation D0288-apps]$ oc new-app \
--name bonjour \
httpd:2.4~https://github.com/${RHT_OCP4_GITHUB_USER}/D0288-apps \
--context-dir s2i-scripts
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "bonjour" created
buildconfig.build.openshift.io "bonjour" created
deployment.apps "bonjour" created
service "bonjour" created
--> Success
...output omitted...
```

- 3.5. Sehen Sie sich die Build-Logs an. Warten Sie, bis der Build abgeschlossen ist und das Anwendungscontainer-Image an die RHOC-P-Registry übertragen wurde:

```
[student@workstation D0288-apps]$ oc logs -f bc/bonjour
Cloning "https://github.com/youruser/D0288-apps" ...
...output omitted...
---> Enabling s2i support in httpd24 image
AllowOverride All
---> Installing application source
---> Creating info page
Pushing image image-registry.openshift-image-registry.svc:5000/youruser-s2i-
scripts/bonjour:latest ... ...
...output omitted...
Push successful
```

Überprüfen Sie, ob anstelle der integrierten S2I-Skripts aus dem Builder-Image die von der Anwendung bereitgestellten, benutzerdefinierten S2I-Skripts verwendet werden.

► 4. Testen Sie die Anwendung.

- 4.1. Warten Sie, bis die Anwendung bereitgestellt wird, und zeigen Sie dann den Status des Anwendungs-Pods an. Der Anwendungs-Pod sollte den Status **Running** aufweisen:

```
[student@workstation D0288-apps]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
bonjour-1-build   0/1     Completed   0          105s
bonjour-7bc86dd97-wjvhx   1/1     Running    0          72s
```

- 4.2. Stellen Sie die Anwendung für den externen Zugriff mit einer Route bereit.

```
[student@workstation D0288-apps]$ oc expose svc bonjour
route.route.openshift.io/bonjour exposed
```

- 4.3. Rufen Sie die Routen-URL mit dem Befehl `oc get route` ab:

```
[student@workstation D0288-apps]$ oc get route
NAME      HOST/PORT
bonjour   bonjour-youruser-s2i-scripts.apps.cluster.domain.example.com ...
```

- 4.4. Rufen Sie mit dem Befehl `curl` und der Routen-URL aus dem vorherigen Befehl die Indexseite der Anwendung auf:

```
[student@workstation D0288-apps]$ curl \
http://bonjour-${RHT_OCP4_DEV_USER}-s2i-scripts.${RHT_OCP4_WILDCARD_DOMAIN}
Hello Class! D0288 rocks!!!
```

Die Inhalte der Datei `index.html` sollten in der Anwendungsquelle angezeigt werden.

- 4.5. Rufen Sie die Informationsseite der Anwendung mit dem Befehl `curl` auf:

```
[student@workstation D0288-apps]$ curl \
http://bonjour-$[RHT_OCP4_DEV_USER]-s2i-scripts.$[RHT_OCP4_WILDCARD_DOMAIN]\
/info.html
Page built on Jun 11, 2021 @ 16:12 PM
Proudly served by Apache HTTP Server version 2.4
```

Die Inhalte der Datei `info.html` mit den Angaben zur Erstellungszeit und Version des Apache HTTP-Servers sollten angezeigt werden.

- 4.6. Überprüfen Sie die Logs für den Anwendungs-Pod. Denken Sie daran, dass der Log-Level für den Startvorgang im Skript `run` in `debug` geändert wurde. Die Logmeldungen auf `debug`-Ebene sollten während des Startvorgangs angezeigt werden:

```
[student@workstation D0288-apps]$ oc logs deployment/bonjour
...output omitted...
[Fri Nov 03 16:12:21.690941 2021] [so:debug] [pid 9] mod_so.c(266): AH01575:
 loaded module systemd_module from /opt/rh/httpd24/root/etc/httpd/modules/
mod_systemd.so
[Fri Nov 03 16:12:21.691050 2021] [so:debug] [pid 9] mod_so.c(266): AH01575:
 loaded module cgi_module from /opt/rh/httpd24/root/etc/httpd/modules/mod_cgi.so
...output omitted...
[Fri Nov 03 16:12:21.742471 2021] [ssl:debug] [pid 9] ssl_engine_init.c(270):
AH01886: SSL FIPS mode disabled
...output omitted...
[Fri Nov 03 16:12:21.745520 2021] [mpm_prefork:notice] [pid 9] AH00163:
Apache/2.4.25 (Red Hat) OpenSSL/1.0.1e-fips configured -- resuming normal
operations
[Fri Nov 03 16:12:21.745530 2021] [core:notice] [pid 9] AH00094: Command line:
'httpd -D FOREGROUND -e debug'
...output omitted...
10.131.0.16 - - [03/Nov/2021:16:28:53 +0000] "GET / HTTP/1.1" 200 28 "-"
"curl/7.29.0"
10.128.2.216 - - [03/Nov/2021:16:29:03 +0000] "GET /info.html HTTP/1.1" 200 87 "-"
"curl/7.29.0"
```

- 5. Bereinigung. Löschen Sie das Projekt.

```
[student@workstation D0288-apps]$ cd ~
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-s2i-scripts
project.project.openshift.io "youruser-redhat-com-s2i-scripts" deleted
```

- 6. Löschen Sie den Container `test`, den Sie zuvor zur Anzeige der S2I-Standardskripts erstellt haben.

```
[student@workstation ~]$ podman rm test
925b932059df9e327bffffe1964c7a1a5a45d13d872b2fb67e333b63166923ce3
```

Ihr ID-Wert unterscheidet sich vom obigen.

Beenden

Führen Sie auf der workstation-VM das Skript `lab s2i-scripts finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken.

```
[student@workstation ~]$ lab s2i-scripts finish

Completing Guided Exercise: Customizing S2I Builds

· Log in on OpenShift..... SUCCESS
· Git repo '/home/student/D0288-apps' has no pending changes.. SUCCESS

Please use start if you wish to do the exercise again.
```

Hiermit ist die angeleitete Übung beendet.

Erstellen eines S2I-Basis-Images

Ziele

Nach Abschluss dieses Abschnitts sollten Sie in der Lage sein, ein S2I-Builder-Image mit dem Befehlszeilentool `s2i` zu erstellen.

Erstellen und Veröffentlichen eines benutzerdefinierten S2I-Builder-Images

Beim S2I-Build-Prozess wird Anwendungsquellcode mit einem geeigneten S2I-Builder-Image kombiniert, um das endgültige Anwendungscontainer-Image zu erzeugen, das auf einem Red Hat OpenShift Container Platform-Cluster (RHOC) bereitgestellt wird.

Es ist wichtig, das S2I-Builder-Image mit dem Befehlszeilentool `s2i` zu erstellen und zu testen, bevor es auf dem RHOC-Cluster bereitgestellt wird und von anderen Entwicklern zum Erstellen von Anwendungen genutzt werden kann. Installieren Sie das Tool `s2i` auf Ihrem lokalen Rechner, um Ihre S2I-Builder-Images außerhalb eines RHOC-Clusters zu erstellen und zu testen.

Installieren des S2I-Tools

Seit RHEL 7 ist das Tool `s2i` im Paket `source-to-image` verfügbar und kann mit Yum installiert werden. Stellen Sie sicher, dass für Ihr System das Yum-Repository `rhel-server-rhscl-7-rpms` oder `rhel-server-rhscl-8-rpms` abonniert und aktiviert wurde (abhängig von Ihrer RHEL-Version).

Bei anderen Betriebssystemen kann das Tool `s2i` auf der Seite des Upstream-Projekts `source-to-image` heruntergeladen werden: <https://github.com/openshift/source-to-image/releases>

Verwenden des S2I-Tools

Führen Sie den Befehl `s2i create` aus, um die erforderlichen Vorlagendateien zur Erstellung eines neuen S2I-Builder-Images zu erstellen:

```
[user@host ~]$ s2i create image_name directory
```

Der obige Befehl erstellt den Ordner `directory` und füllt ihn mit den folgenden Vorlagendateien, die Sie bei Bedarf aktualisieren können:

```
directory
├── Dockerfile ①
├── Makefile
├── README.md
└── s2i ②
    └── bin
        ├── assemble
        ├── run
        ├── save-artifacts
        └── usage
```

```
└── test
    ├── run
    └── test-app ③
        └── index.html
```

- ① Das Dockerfile für das S2I-Builder-Image
- ② Das S2I-Skriptverzeichnis
- ③ Ordner, in den Sie den Anwendungsquellcode für lokale Tests kopieren

Mit dem Befehl `s2i create` wird eine Dockerfile-Vorlage mit Kommentaren erstellt, die speziell für die Ausführung auf einem RHOCOP-Cluster mit einer zufälligen Benutzer-ID und OpenShift-spezifischen Bezeichnungen angepasst ist. Darüber hinaus werden Stubs für die S2I-Skripts erstellt, die Sie für die Anforderungen Ihrer Anwendung anpassen können. Nachdem Sie das Dockerfile und die S2I-Skripts wie gewünscht aktualisiert haben, können Sie das Builder-Image mit dem Befehl `podman build` erstellen.

```
[user@host ~]$ podman build -t builder_image .
```

Wenn das Builder-Image bereit zur Verwendung ist, können Sie mit dem Befehl `s2i build` ein Anwendungscontainer-Image erstellen. Dies ermöglicht Ihnen, das S2I-Builder-Image lokal zu testen, ohne dass Sie es an einen Registry-Server übertragen und eine Anwendung mit dem Builder-Image in einem RHOCOP-Cluster bereitzustellen müssen:

```
[user@host ~]$ s2i build src builder_image tag_name
```



Anmerkung

Wenn Sie Anweisungen, die nicht Teil der OCI-Spezifikation sind, wie `ONBUILD`, in Ihr Builder-Image-Dockerfile aufnehmen, müssen Sie beim Erstellen Ihres S2I-Builder-Images die Option `--format docker` mit dem Befehl `podman build` verwenden. Diese Option überschreibt das von Podman verwendete `oci`-Standardformat.

Der Befehl `s2i build` kombiniert den in der Option `src` definierten Anwendungsquellcode mit dem Container-Image `builder_image`, um das Anwendungscontainer-Image mit dem Tag `tag_name` zu erzeugen. Dieser Befehl emuliert den S2I-Prozess, den der RHOCOP-Cluster verwendet, indem der bereitgestellte Quellcode automatisch in das Builder-Image eingefügt wird, aber er verwendet den lokalen Docker-Service, um ein Testcontainer-Image zu erstellen, anstatt das Image in RHOCOP bereitzustellen.

Testen Sie das vom Befehl `s2i build` erstellte Anwendungscontainer-Image, indem Sie das Image mit dem Befehl `podman run` ausführen. Beachten Sie, dass Sie bei einer Bereitstellung des Anwendungscontainer-Images in RHOCOP die Ausführung des Containers mit einer zufälligen Benutzer-ID und dem Flag `-u` für den Befehl `podman run` simulieren müssen.



Wichtig

Für den Befehl `s2i build` ist die Verwendung eines lokalen Docker-Service erforderlich, da er die Docker-API direkt über den Socket verwendet, um das S2I-Container-Image zu erstellen. In RHEL 8- und RHOCOP 4-Umgebungen ist Docker nicht enthalten und dies funktioniert nicht.

Um Unterstützung für Umgebungen bereitzustellen, für die Docker nicht verfügbar ist, enthält der Befehl `s2i build` jetzt die Option `--as-dockerfile path/to/Dockerfile`. Mit dieser Option wird der Befehl `s2i build` so konfiguriert, dass ein Dockerfile und zwei unterstützende Verzeichnisse erstellt werden, mit denen Sie mit dem Befehl `podman build` ein Testcontainer-Image aus einem Quell-Repository und einem Builder-Image erstellen können. Wenn Sie diese Option verwenden, ist kein lokaler Docker-Daemon erforderlich, um den Befehl `s2i build` auszuführen.

Sie können den Speicherort eines Verzeichnisses oder eine Git-Repository-URL angeben, die die Anwendungsquelle für die Option `src` enthält.

Sie müssen für inkrementelle Builds ein `save-artifacts`-Skript erstellen und das Flag `--incremental` an den Befehl `s2i build` übergeben. Wenn ein `save-artifacts`-Skript und ein vorheriges Image vorhanden sind und Sie die Option `--incremental=true` verwenden, dann lautet der Workflow wie folgt:

1. S2I erstellt ein neues Container-Image aus dem vorherigen Build-Image.
2. S2I führt das Skript `save-artifacts` in diesem Container aus. Dieses Skript ist dafür zuständig, eine tar-Datei der Artefakte an `stdout` zu streamen.
3. S2I erstellt das neue Ausgabe-Image:
 - a. Die Artefakte aus dem vorherigen Build befinden sich im Verzeichnis `artifacts` der tar-Datei, die an den Build übergeben wurde.
 - b. Das Skript `assemble` des S2I-Builder-Images ist für die Erkennung und Verwendung der Build-Artefakte zuständig.

Führen Sie die Befehle `s2i --help` und `s2i subcommand --help` aus, um die verschiedenen Sub-Befehle, die zugehörigen Optionen und Verwendungsbeispiele anzuzeigen.

Nachdem Sie das Anwendungscontainer-Image lokal getestet haben, können Sie das S2I-Builder-Image in eine Registry kopieren. Sie müssen vor der Bereitstellung von Anwendungen auf einem RHOCOP-Cluster mit dem Builder-Image mit dem Befehl `oc import-image` einen Image-Stream erstellen, der auf dem Builder-Image basiert. Sie können anschließend den Image-Stream verwenden, um Anwendungen in RHOCOP zu erstellen.

Erstellen eines Nginx-S2I-Builder-Images

Führen Sie die folgenden Schritte aus, um ein Nginx-S2I-Builder-Image auf Basis von RHEL 8 zu erstellen:

Erstellen des Dockerfile-Projekts und Auffüllen des Projekts mit Daten

Erstellen Sie mit dem Befehl `s2i` das Projektverzeichnis für das S2I-Builder-Image und bearbeiten Sie bei Bedarf die Dateien:

- Führen Sie den Befehl `s2i create` aus, um die Struktur des Skeleton-Verzeichnis für Artefakte des S2I-Builder-Images zu erstellen:

```
[user@host ~]$ s2i create s2i-do288-nginx s2i-do288-nginx
```

- Bearbeiten Sie das Dockerfile, um Anweisungen zur Installation von Nginx und der S2I-Skripts hinzuzufügen. Das folgende Dockerfile für ein Nginx-S2I-Builder-Image verwendet das RHEL 8 Universal Base Image:

```
FROM registry.access.redhat.com/ubi8/ubi:8.0 1

ENV X_SCLS="rh-nginx18" \
    PATH="/opt/rh/rh-nginx18/root/usr/sbin:$PATH" \
    NGINX_DOCROOT="/opt/rh/rh-nginx18/root/usr/share/nginx/html"

LABEL io.k8s.description="A Nginx S2I builder image" \
      2
      io.k8s.display-name="Nginx 1.8 S2I builder image for DO288" \
      io.openshift.expose-services="8080:http" \
      io.openshift.s2i.scripts-url="image:///usr/libexec/s2i" \
      io.openshift.tags="builder,webserver,nginx,nginx18,html"

ADD nginxconf.sed /tmp/
COPY ./s2i/bin/ /usr/libexec/s2i 3

RUN yum install -y --nodocs rh-nginx18 \
    && yum clean all \
    && sed -i -f /tmp/nginxconf.sed /etc/opt/rh/rh-nginx18/nginx/nginx.conf \
    && chgrp -R 0 /var/opt/rh/rh-nginx18 /opt/rh/rh-nginx18 \
5      && chmod -R g=u /var/opt/rh/rh-nginx18 /opt/rh/rh-nginx18 \
6      && echo 'Hello from the Nginx S2I builder image' > ${NGINX_DOCROOT}/index.html

EXPOSE 8080

USER 1001

CMD ["/usr/libexec/s2i/usage"]
```

- ① Verwenden Sie das RHEL 8 Universal Base Image als Basis für das S2I-Builder-Image.
 - ② Bezeichnungsmetadaten für Nutzer von S2I-Builder-Images.
 - ③ Kopieren Sie die S2I-Skripts an den Speicherort, der durch die Bezeichnung `io.openshift.s2i.scripts-url` angegeben wird.
 - ④ Installieren Sie Nginx.
 - ⑤ ⑥ Legen Sie Berechtigungen für die Ausführung mit einer zufälligen Benutzer-ID auf einem RHOC-Cluster fest.
- Erstellen Sie im Verzeichnis `.s2i/bin` der Anwendungsquelle ein `assemble`-Skript mit dem folgenden Inhalt, das die HTML-Quelldateien in das Dokumentstammverzeichnis des Nginx-Webservers kopiert:

```
#!/bin/bash -e

echo "----> Copying source HTML files to web server root..."
cp -Rf /tmp/src/. /opt/rh/rh-nginx18/root/usr/share/nginx/html/
```

- Erstellen Sie im Verzeichnis `.s2i/bin` der Anwendungsquelle ein `run`-Skript mit dem folgenden Inhalt, das den Nginx-Webserver im Vordergrund ausführt:

```
#!/bin/bash -e

exec nginx -g "daemon off;"
```

Erstellen und Testen des S2I-Builder-Images

Erstellen Sie mit Podman und dem Befehl `s2i` das S2I-Builder-Image und ein Container-Image für die Testanwendung:

- Erstellen Sie das S2I-Builder-Image:

```
[user@host ~]$ podman build -t s2i-do288-nginx .
```

- Führen Sie den Befehl `s2i build` aus, um ein Container-Image für die Testanwendung zu erstellen. Zum Überschreiben der im Builder-Image enthaltenen Standarddatei `index.html` erstellen Sie im Verzeichnis `test/test-app` eine `index.html`-Datei, um diese neue Datei in den Nginx-Testcontainer einzufügen:

```
[user@host ~]$ s2i build test/test-app s2i-do288-nginx nginx-test \
--as-docker-file /path/to/Dockerfile
```

- Um den Testcontainer zu erzeugen, verwenden Sie den Befehl `podman build`, dieses Mal mit dem vom Befehl `s2i build` generierten Dockerfile:

```
[user@host ~]$ podman build -t nginx-test /path/to/Dockerfile
```

- Um das Container-Image zu testen, führen Sie den Befehl `podman run` mit einer anderen Benutzer-ID aus als der ID, die im Dockerfile angegeben ist. So stellen Sie sicher, dass der Container mit einer zufällig erstellten Benutzer-ID auf einem RHOC-Cluster ausgeführt werden kann:

```
[user@host ~]$ podman run -u 1234 -d -p 8080:8080 nginx-test
```

Wenn der Container fehlerfrei ausgeführt, überprüfen Sie, ob die Testdatei `index.html`, die Sie im Verzeichnis `test` bereitgestellt haben, beim Testen des Nginx-Containers gerendert wird.

Bereitstellen des S2I-Builder-Images in RHOC

Übertragen Sie mit dem Befehl `skopeo` das S2I-Builder-Image an eine Container-Registry und erstellen Sie einen Image-Stream in RHOC, der auf dieses Image verweist.

- Nachdem Sie den Container lokal getestet haben, übertragen Sie das S2I-Builder-Image an eine Unternehmens-Registry. Angenommen, Sie haben ein Quay.io-Benutzerkonto und haben sich mit dem Befehl `podman login` angemeldet:

```
[user@host ~]$ skopeo copy containers-storage:localhost/s2i-do288-httdp \
docker://quay.io/${RHT_OCP4_QUAY_USER}/s2i-do288-httdp
```

- Um einen Image-Stream für das Nginx-S2I-Builder-Image zu erstellen, erstellen Sie ein neues Projekt und führen den Befehl `oc import-image` aus:

```
[user@host ~]$ oc import-image s2i-do288-nginx \
--from quay.io/${RHT_OCP4_QUAY_USER}/s2i-do288-nginx \
--confirm
```



Anmerkung

Denken Sie beim Kapitel 3, *Veröffentlichen von Enterprise Container-Images* daran, dass Sie möglicherweise ein Secret zum Abrufen mit Ihren Anmeldedaten für die Remote-Registry erstellen müssen, in der Sie Ihr S2I-Builder-Image veröffentlichen, wenn dieses Image nicht öffentlich verfügbar ist. Sie müssen dieses Secret auch mit dem Standardservicekonto verknüpfen, das zum Abrufen von Images verwendet wird, um den Image-Stream mit dem Befehl `oc import-image` zu erstellen.

- Nach Erstellung des Image-Streams können Sie diesen verwenden, um mit dem Nginx-S2I-Builder-Image Anwendungen zu erstellen. Beachten Sie, dass Sie die Tilde-Notation verwenden müssen, es sei denn, Ihr S2I-Builder-Image stellt eine Alternative zu den vom RHOCB-Befehl `oc new-app` unterstützten Sprachen dar:

```
[user@host ~]$ oc new-app --name nginx-test \
s2i-do288-nginx~git_repository
```



Literaturhinweise

Weitere Informationen finden Sie im Kapitel *Creating Images* des Handbuchs *Images* für Red Hat RHOCB 4.6 unter
https://access.redhat.com/documentation/en-us/red_hat_openshift_container_platform/4.6/html-single/images/creating-images#creating-images

Erstellen eines S2I Builder Image

<https://blog.openshift.com/create-s2i-builder-image>

Source-to-Image (S2I) Tool

<https://github.com/openshift/source-to-image>

s2i-Tool Unterbefehl-Referenz

<https://github.com/openshift/source-to-image/blob/master/docs/cli.md>

► Angeleitete Übung

Erstellen eines S2I-Basis-Images

In dieser Übung erstellen und testen Sie ein S2I-Builder-Image des Apache HTTP-Servers. Anschließend erstellen Sie eine Anwendung anhand des Images und stellen diese bereit.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Erstellen eines S2I-Builder-Image für den Apache HTTP-Server mit dem Tool `s2i`
- Lokales Testen des S2I-Builder-Images mit einer einfachen Anwendung
- Veröffentlichen des Builder-Images in der Quay.io-Container-Registry
- Bereitstellen und Testen einer Anwendung auf einem Red Hat OpenShift Container Platform(RHOC)P-Cluster mit dem S2I-Builder-Image

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchzuführen zu können:

- Auf einen aktiven RHOC-Cluster
- Auf das übergeordnete Image (`ubi8/ubi`) für die Beispieldaten
- Auf die Anwendung `html-helloworld` im Git-Repository `D0288-apps`.

Führen Sie den folgenden Befehl auf der `workstation`-VM aus, um die Voraussetzungen für die Übung zu überprüfen sowie die Übungs- und Lösungsdateien herunterzuladen:

```
[student@workstation ~]$ lab apache-s2i start
```

Anweisungen

- 1. Überprüfen Sie auf der `workstation`-VM, ob das Paket `source-to-image` installiert ist, mit dem das Befehlszeilentool `s2i` bereitgestellt wird:

```
[student@workstation ~]$ s2i version
s2i v1.3.1
```

- 2. Erstellen Sie mit dem Befehl `s2i` die Vorlagendateien und Verzeichnisse, die für das S2I-Builder-Image benötigt werden.

- 2.1. Erstellen Sie auf der `workstation`-VM im Verzeichnis `/home/student/` mit dem Befehl `s2i create` die Vorlagendateien für das Builder-Image:

```
[student@workstation ~]$ s2i create s2i-d0288-httpd s2i-d0288-httpd
```

- 2.2. Überprüfen Sie, ob die Vorlagendateien erstellt wurden. Mit dem Befehl `s2i create` wird die folgende Verzeichnisstruktur erstellt:

```
[student@workstation ~]$ tree s2i-do288-httdp
s2i-do288-httdp
├── Dockerfile
├── Makefile
└── README.md
└── s2i
    ├── bin
    │   ├── assemble
    │   ├── run
    │   ├── save-artifacts
    │   └── usage
    └── test
        ├── run
        └── test-app
            └── index.html
4 directories, 9 files
```



Anmerkung

Der Befehl „`s2i`“ generiert ein *Containerfile* mit dem Legacy-Namen *Dockerfile*. „*Containerfile*“ ist der bevorzugte zu verwendende Name.

- 3. Erstellen Sie das S2I-Builder-Image für den Apache HTTP-Server.

- 3.1. Ein Beispiel-Containerfile für das Builder-Image des Apache HTTP-Servers finden Sie unter `~/D0288/labs/apache-s2i/Containerfile`. Überprüfen Sie diese Datei kurz:

```
[student@workstation ~]$ cat ~/D0288/labs/apache-s2i/Containerfile
FROM registry.access.redhat.com/ubi8/ubi:8.4 ①

# Generic labels
LABEL Component="httdp" \
      Name="s2i-do288-httdp" \
      Version="1.0" \
      Release="1"

# Labels consumed by RHOC
LABEL io.k8s.description="A basic Apache HTTP Server S2I builder image" \
      io.k8s.display-name="Apache HTTP Server S2I builder image for D0288" \
      io.openshift.expose-services="8080:http" \
      io.openshift.s2i.scripts-url="image:///usr/libexec/s2i" ② ③

# This label is used to categorize this image as a builder image in the
# RHOC web console.
LABEL io.openshift.tags="builder, httdp, httdp24"

# Apache HTTP Server DocRoot
ENV DOCROOT /var/www/html
```

Kapitel 5 | Anpassen von Source-to-Image-Builds

```
RUN yum install -y --nодocs --disableplugin=subscription-manager httpd && \
    yum clean all --disableplugin=subscription-manager -y && \
    echo "This is the default index page from the s2i-do288-httpd S2I builder
image." > ${DOCROOT}/index.html ④

# Change web server port to 8080
RUN sed -i "s/Listen 80/Listen 8080/g" /etc/httpd/conf/httpd.conf

# Copy the S2I scripts to the default location indicated by the
# io.openshift.s2i.scripts-url LABEL (default is /usr/libexec/s2i)
COPY ./s2i/bin/ /usr/libexec/s2i ⑥
...output omitted...
```

- ①** Verwenden Sie das RHEL 8 Universal Base Image als Basis für diesen Container.
- ②** Legen Sie die Bezeichnungen fest, die für RHOCP verwendet werden, um das Builder-Image zu beschreiben.
- ③** Konfigurieren Sie, wo sich die obligatorischen S2I-Skripts (`run`, `assemble`) befinden.
- ④** Installieren Sie das httpd-Webserverpaket und bereinigen Sie den Yum-Cache.
- ⑤** Legen Sie den Inhalt der Standarddatei `index.html` für das Builder-Image fest.
- ⑥** Kopieren Sie die S2I-Skripts in das Verzeichnis `/usr/libexec/s2i`.

Kopieren Sie dann dieses Containerfile in das Verzeichnis `~/s2i-do288-httpd`, und überschreiben Sie das generierte Vorlagen-Containerfile:

```
[student@workstation ~]$ rm ~/s2i-do288-httpd/Dockerfile
[student@workstation ~]$ cp ~/D0288/labs/apache-s2i/Containerfile
~/s2i-do288-httpd/
```

- 3.2. Überprüfen und kopieren Sie in ähnlicher Weise die S2I-Skripts für dieses Builder-Image aus dem Verzeichnis `~/D0288/labs/apache-s2i/s2i/bin` in das Verzeichnis `~/s2i-do288-httpd/s2i/bin`, und überschreiben Sie die generierten Skripts:

```
[student@workstation ~]$ cp -Rv ~/D0288/labs/apache-s2i/s2i ~/s2i-do288-httpd/
'D0288/labs/apache-s2i/s2i/bin/assemble' -> 's2i-do288-httpd/s2i/bin/assemble'
'D0288/labs/apache-s2i/s2i/bin/usage' -> 's2i-do288-httpd/s2i/bin/usage'
'D0288/labs/apache-s2i/s2i/bin/run' -> 's2i-do288-httpd/s2i/bin/run'
```

- 3.3. In dieser Übung wird das Skript `save-artifacts` nicht implementiert. Entfernen Sie es aus dem Verzeichnis `~/s2i-do288-httpd/s2i/bin`:

```
[student@workstation ~]$ rm -f ~/s2i-do288-httpd/s2i/bin/save-artifacts
```

- 3.4. Erstellen Sie das S2I-Builder-Image für den Apache HTTP-Server. Vergessen Sie nicht den Punkt am Ende des Befehls `podman build`. Dieser weist Podman an, das Image anhand des Containerfiles im aktuellen Verzeichnis zu erstellen:

```
[student@workstation ~]$ cd s2i-do288-httdp  
[student@workstation s2i-do288-httdp]$ podman build -t s2i-do288-httdp .  
STEP 1: FROM registry.access.redhat.com/ubi8/ubi:8.4  
...output omitted...  
STEP 14: COMMIT s2i-do288-httdp  
...output omitted...
```

3.5. Überprüfen Sie, ob das Builder-Image erstellt wurde:

```
[student@workstation s2i-do288-httdp]$ podman images  
REPOSITORY                TAG      IMAGE ID      CREATED  
localhost/s2i-do288-httdp    latest   82beb27428b7  9 seconds ago  
registry.access.redhat.com/ubi8/ubi     8.4      7ae69d957d8b  2 weeks ago  
...output omitted...
```

- 4. Erstellen und testen Sie ein Anwendungscontainer-Image, das das S2I-Builder-Image für den Apache HTTP-Server und den Anwendungsquellcode kombiniert.

- 4.1. Wenn das Builder-Image fertiggestellt ist, können Sie mit dem Befehl `s2i build` das Anwendungscontainer-Image erstellen. Sehen Sie sich vor der Erstellung des Anwendungscontainer-Images die HTML-Beispieldatei `~/D0288/labs/apache-s2i/index.html` an:

```
[student@workstation s2i-do288-httdp]$ cat ~/D0288/labs/apache-s2i/index.html  
This is the index page from the app
```

Kopieren Sie dann diese Datei in das Verzeichnis `~/s2i-do288-httdp/test/test-app`, um die im Builder-Image gepackte Datei `index.html` zu überschreiben:

```
[student@workstation s2i-do288-httdp]$ cp ~/D0288/labs/apache-s2i/index.html \  
~/s2i-do288-httdp/test/test-app/
```

- 4.2. Erstellen Sie ein neues Verzeichnis für das durch den Befehl `s2i build` generierte Containerfile.

```
[student@workstation s2i-do288-httdp]$ mkdir ~/s2i-sample-app
```

- 4.3. Erstellen Sie das Anwendungscontainer-Image:

```
[student@workstation s2i-do288-httdp]$ s2i build test/test-app/ \  
s2i-do288-httdp s2i-sample-app \  
--as-dockerfile ~/s2i-sample-app/Containerfile  
Application dockerfile generated in /home/student/s2i-sample-app/Containerfile
```

- 4.4. Überprüfen Sie das generierte Anwendungsverzeichnis.

```
[student@workstation s2i-do288-httdp]$ cd ~/s2i-sample-app  
[student@workstation s2i-sample-app]$ tree .  
.  
└── Containerfile
```

Kapitel 5 | Anpassen von Source-to-Image-Builds

```
└── downloads
    └── defaultScripts
        └── scripts
    └── upload
        └── scripts
            └── src
                └── index.html
```

6 directories, 2 files

Beachten Sie die Datei `index.html` im Verzeichnis `upload/src`. Dies ist die gleiche `index.html`-Datei, die Sie in einem vorherigen Schritt in das Verzeichnis `test/test-app` kopiert haben.

4.5. Überprüfen Sie das generierte Containerfile.

```
[student@workstation s2i-sample-app]$ cat Containerfile
FROM s2i-do288-httdp ①
LABEL "io.k8s.display-name"="s2i-sample-app" \
      "io.openshift.s2i.build.image"="s2i-do288-httdp" \
      "io.openshift.s2i.build.source-location"="test/test-app/"

USER root
# Copying in source code
COPY upload/src /tmp/src ③
# Change file ownership to the assemble user. Builder image must support chown
# command.
RUN chown -R 1001:0 /tmp/src
USER 1001
# Assemble script sourced from builder image based on user input or image
# metadata.
# If this file does not exist in the image, the build will fail.
RUN /usr/libexec/s2i/assemble
# Run script sourced from builder image based on user input or image metadata.
# If this file does not exist in the image, the build will fail.
CMD /usr/libexec/s2i/run
```

- ① Verwenden Sie das Builder-Image `s2i-do288-httdp` als übergeordnetes Element dieses Container-Images.
- ② Legen Sie die Bezeichnungen fest, die für RHOCP verwendet werden, um die Anwendung zu beschreiben.
- ③ Kopieren Sie den Quellcode, der im Verzeichnis `upload/src` enthalten ist.

4.6. Erstellen Sie anhand des generierten Containerfiles ein Testcontainer-Image.

```
[student@workstation s2i-sample-app]$ podman build \
-t s2i-sample-app .
STEP 1: FROM s2i-do288-httdp
STEP 2: LABEL "io.k8s.display-name"="s2i-sample-app"...output omitted...
...output omitted...
STEP 9: COMMIT s2i-sample-app
...output omitted...
```

- 4.7. Überprüfen Sie, ob das Anwendungscontainer-Image erstellt wurde:

```
[student@workstation s2i-sample-app]$ podman images
REPOSITORY                                     TAG      IMAGE ID      CREATED
localhost/s2i-sample-app                      latest   3c8637c4372d  About an hour ago
localhost/s2i-do288-httdp                     latest   d06040a9eeca  About an hour ago
...output omitted...
```

- 4.8. Testen Sie das Anwendungscontainer-Image lokal auf der workstation-VM. Führen Sie den Container als zufälliger Benutzer mit dem Flag -u aus, um eine Ausführung durch einen zufälligen Benutzer auf einem RHOC-Cluster zu simulieren. Sie können den Befehl kopieren oder direkt aus dem Skript ~/DO288/labs/apache-s2i/local-test.sh ausführen:

```
[student@workstation s2i-sample-app]$ podman run --name test -u 1234 \
-p 8080:8080 -d s2i-sample-app
a5f4b3e5bf ...output omitted...
```

- 4.9. Überprüfen Sie, ob der Container gestartet wurde:

```
[student@workstation s2i-sample-app]$ podman ps
CONTAINER ID   IMAGE                               COMMAND ...
a5f4b3e5bfaa   localhost/s2i-sample-app:latest   /bin/sh -c /usr/lib...
...output omitted...
```

- 4.10. Führen Sie den Befehl curl aus, um die Anwendung zu testen:

```
[student@workstation s2i-sample-app]$ curl http://localhost:8080
This is the index page from the app
```

- 4.11. Beenden Sie den Test-Anwendungscontainer:

```
[student@workstation s2i-sample-app]$ podman stop test
a5f4b3e5bf ...output omitted...
```

- 5. Übertragen Sie das S2I-Builder-Image für den Apache HTTP-Server in Ihr Quay.io-Benutzerkonto.

- 5.1. Laden Sie die Konfiguration Ihrer Kursumgebung.

Führen Sie den folgenden Befehl aus, um die in der ersten angeleiteten Übung erstellten Umgebungsvariablen zu laden:

```
[student@workstation s2i-sample-app]$ source /usr/local/etc/ocp4.config
```

- 5.2. Melden Sie sich mit dem Befehl podman bei Ihrem Quay.io-Benutzerkonto an. Sie werden aufgefordert, Ihr Passwort einzugeben.

```
[student@workstation s2i-sample-app]$ podman login \
-u ${RHT_OCP4_QUAY_USER} quay.io
Password:
Login Succeeded!
```

- 5.3. Veröffentlichen Sie mit dem Befehl skopeo copy das S2I-Builder-Image in Ihrem Quay.io-Benutzerkonto.

```
[student@workstation s2i-sample-app]$ skopeo copy \
containers-storage:localhost/s2i-do288-httd \
docker://quay.io/${RHT_OCP4_QUAY_USER}/s2i-do288-httd
...output omitted...
Writing manifest to image destination
Storing signatures
```

▶ 6. Erstellen Sie einen Image-Stream für das S2I-Builder-Image des Apache HTTP-Servers.

- 6.1. Melden Sie sich bei RHOCP an, und erstellen Sie ein neues Projekt. Stellen Sie dem Projektnamen Ihren Entwicklerbenutzernamen voran.

```
[student@workstation s2i-sample-app]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
[student@workstation s2i-sample-app]$ oc new-project \
${RHT_OCP4_DEV_USER}-apache-s2i
Now using project "youruser-apache-s2i" on server "https://
api.cluster.domain.example.com:6443".
...output omitted...
```

- 6.2. Erstellen Sie ein Secret aus dem Zugriffstoken der Container-Registry-API, das von Podman gespeichert wurde.

Sie können auch den folgenden oc create secret-Befehl aus dem Skript create-secret.sh ausführen oder ausschneiden und in das Verzeichnis /home/student/D0288/labs/apache-s2i einfügen.

```
[student@workstation s2i-sample-app]$ oc create secret generic quayio \
--from-file .dockerconfigjson=${XDG_RUNTIME_DIR}/containers/auth.json \
--type=kubernetes.io/dockerconfigjson
secret/quayio created
```

- 6.3. Verknüpfen Sie das neue Secret mit dem Servicekonto builder.

```
[student@workstation s2i-sample-app]$ oc secrets link builder quayio
```

- 6.4. Erstellen Sie einen Image-Stream, indem Sie das S2I-Builder-Image aus der Quay.io-Container-Registry importieren:

```
[student@workstation s2i-sample-app]$ oc import-image s2i-do288-httd \ 
--from quay.io/${RHT_OCP4_QUAY_USER}/s2i-do288-httd --confirm
imagestream.image.openshift.io/s2i-do288-httd imported

Name: s2i-do288-httd
Namespace: youruser-apache-s2i
Created: Less than a second ago
Labels: <none>
Annotations: openshift.io/image.dockerRepositoryCheck=2019-06-26T02:30:59Z
Image Repository: image-registry.openshift-image-registry.svc:5000/youruser-
apache-s2i/s2i-do288-httd
Image Lookup: local=false
Unique Images: 1
Tags: 1

latest
tagged from quay.io/youruser/s2i-do288-httd

* quay.io/youruser/s2i-do288-httd@sha256:fe0cd09432...
  Less than a second ago

...output omitted...
```

6.5. Überprüfen Sie, ob der Image-Stream erstellt wurde:

```
[student@workstation s2i-sample-app]$ oc get is
NAME          IMAGE REPOSITORY      ...output omitted...
s2i-do288-httd  ...youruser-apache-s2i/s2i-do288-httd
```

- 7. Stellen Sie aus dem Git-Repository des Kursraums die Anwendung `html-helloworld` auf einem RHOC-Cluster bereit, und testen Sie sie. Diese Anwendung besteht aus einer einzelnen HTML-Datei, die eine Meldung anzeigt.
- 7.1. Erstellen Sie eine neue Anwendung mit dem Namen `hello` anhand von Quellen in Git. Sie müssen der Git-URL den Image-Stream `s2i-do288-httd` als Präfix voranstellen. So stellen Sie sicher, dass die Anwendung das zuvor erstellte Builder-Image für den Apache HTTP-Server verwendet:

```
[student@workstation s2i-sample-app]$ oc new-app --name hello-s2i \
s2i-do288-httd-https://github.com/${RHT_OCP4_GITHUB_USER}/D0288-apps \
--context-dir=html-helloworld
--> Found image c7a496d (2 hours old) in image stream "youruser-apache-s2i/s2i-
do288-httd" under tag "latest" for "s2i-do288-httd"

Apache HTTP Server S2I builder image for D0288
-----
A basic Apache HTTP Server S2I builder image
...output omitted...
--> Creating resources ...
...output omitted...
--> Success
...output omitted...
```

Kapitel 5 | Anpassen von Source-to-Image-Builds

- 7.2. Sehen Sie sich die Build-Logs an. Warten Sie, bis der Build abgeschlossen ist und das Anwendungscontainer-Image an die RHOCOP-Registry übertragen wurde:

```
[student@workstation s2i-sample-app]$ oc logs -f bc/hello-s2i
Cloning "https://github.com/youruser/D0288-apps" ...
...output omitted...
---> Copying source files to web server directory...
Pushing image-registry.openshift-image-registry.svc:5000/youruser-apache-s2i/
hello-s2i:latest ...
...output omitted...
Push successful
```

- 7.3. Warten Sie, bis die Anwendung bereitgestellt wird. Zeigen Sie den Status des Anwendungs-Pods an. Der Anwendungs-Pod sollte den Status Running aufweisen:

```
[student@workstation s2i-sample-app]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
hello-s2i-1-build   0/1     Completed   0          71s
hello-s2i-57cb8dc96c-rnkgt   1/1     Running   0          50s
```

- 7.4. Stellen Sie die Anwendung für den externen Zugriff mit einer Route bereit:

```
[student@workstation s2i-sample-app]$ oc expose svc hello-s2i
route.route.openshift.io/hello-s2i exposed
```

- 7.5. Rufen Sie die Routen-URL mit dem Befehl `oc get route` ab:

```
[student@workstation s2i-sample-app]$ export APP_URL=$( \
oc get route/hello-s2i \
-o jsonpath='{.spec.host}{"\n"}')
[student@workstation s2i-sample-app]$ echo ${APP_URL}
hello-s2i-youruser-apache-s2i.apps.cluster.domain.example.com
```

- 7.6. Testen Sie die Anwendung mit der Routen-URL, die Sie im vorherigen Schritt abgerufen haben:

```
[student@workstation s2i-sample-app]$ curl ${APP_URL}
<html>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

- 8. Führen Sie eine Bereinigung durch.

- 8.1. Löschen Sie das Projekt apache-s2i in RHOCOP:

```
[student@workstation s2i-sample-app]$ oc delete project \
${RHT_OCP4_DEV_USER}-apache-s2i
```

- 8.2. Löschen Sie den `test`-Container, der zuvor bei der lokalen Überprüfung der Anwendung erstellt wurde:

```
[student@workstation s2i-sample-app]$ podman rm test  
a5f4b3e5bf ...output omitted...
```

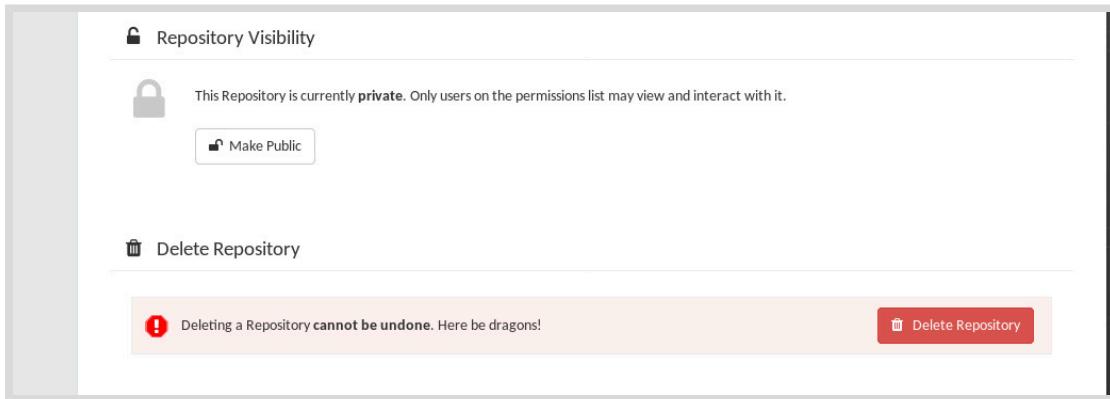
- 8.3. Löschen Sie die Container-Images auf der `workstation`-VM:

```
[student@workstation s2i-sample-app]$ podman rmi -f \  
localhost/s2i-sample-app \  
localhost/s2i-do288-httdp \  
registry.access.redhat.com/ubi8/ubi:8.4  
...output omitted...  
Untagged: localhost/s2i-sample-app:latest  
...output omitted...  
Untagged: localhost/s2i-do288-httdp:latest  
...output omitted...  
Untagged: registry.access.redhat.com/ubi8/ubi:8.4
```

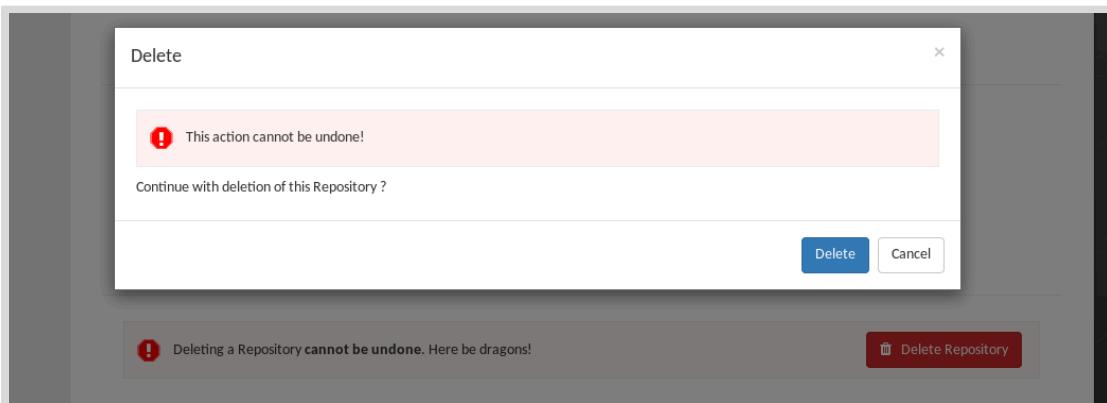
- 8.4. Löschen Sie das Image `s2i-do288-httdp` aus der externen Registry:

```
[student@workstation s2i-sample-app]$ skopeo delete \  
docker://quay.io/${RHT_OCP4_QUAY_USER}/s2i-do288-httdp:latest
```

- 8.5. Melden Sie sich bei Quay.io mit Ihrem persönlichen kostenlosen Benutzerkonto an. Navigieren Sie zu `http://quay.io` und klicken Sie auf **Sign In**, um Ihre Benutzeranmeldedaten einzugeben.
- 8.6. Klicken Sie im Hauptmenü von Quay.io auf **Repositories** und suchen Sie nach `s2i-do288-httdp`. Das Schlosssymbol daneben zeigt an, dass es sich um ein privates Repository handelt, das eine Authentifizierung für Abrufe und Übertragungen erfordert. Klicken Sie auf `s2i-do288-httdp`, um die Seite **Repository Activity** anzuzeigen.
- 8.7. Scrollen Sie auf der Seite **Repository Activity** für das `s2i-do288-httdp`-Repository nach unten, und klicken Sie auf das Zahnradssymbol, um den Tab „Settings“ anzuzeigen. Scrollen Sie nach unten und klicken Sie auf **Delete Repository**.



- 8.8. Klicken Sie im Dialogfeld **Delete** auf **Delete**, um das Löschen des s2i-do288-
httpd-Repository zu bestätigen. Nach einigen Augenblicken werden Sie zur Seite
Repositories zurückgeleitet. Sie können sich jetzt bei Quay.io abmelden.



Beenden

Führen Sie auf der `workstation`-VM das Skript `lab apache-s2i finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken.

```
[student@workstation ~]$ lab apache-s2i finish
```

Hiermit ist die angeleitete Übung beendet.

► Praktische Übung

Anpassen von Source-to-Image-Builds

In dieser praktischen Übung erstellen Sie ein S2I-Builder-Image zur Ausführung von Anwendungen basierend auf der Programmiersprache „Go“.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Erstellen eines S2I-Builder-Images mit der Programmiersprache „Go“, basierend auf dem Universal Base Image für RHEL 8
- Lokales Testen des S2I-Builder-Images durch Erstellen eines Dockerfiles mit dem Tool `s2i` und anschließendem Erzeugen und Ausführen des resultierenden Container-Images mit Podman
- Veröffentlichen des Builder-Images in Ihrem persönlichen Quay.io-Benutzerkonto
- Verwenden des S2I-Builder-Images für Bereitstellung und Test einer Anwendung auf einem Red Hat OpenShift Container Platform(RHOCOP)-Cluster
- Anpassen des Skripts `run` zum Ändern des Verhaltens der Anwendung und erneutem Bereitstellen der Anwendung zum Testen der Änderungen

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um dieses Lab durchzuführen zu können:

- Auf einen aktiven RHOCOP-Cluster
- Auf das Befehlszeilentool `s2i`.
- Auf einen Fork im Git-Repository, der den Quellcode der Anwendung `go-hello` enthält

Führen Sie den folgenden Befehl auf der `workstation`-VM aus, um die Voraussetzungen zu überprüfen. Dieser Befehl lädt zudem die Hilfs- und Lösungsdateien für die Wiederholungsübung herunter:

```
[student@workstation ~]$ lab custom-s2i start
```

Das Setup-Skript erstellt ein Verzeichnis namens `custom-s2i` im Verzeichnis `/home/student/D0288/labs`. Dieser Ordner enthält die Vorlagenordnerstruktur und die Dateien, die mit dem Befehl `s2i create` erstellt wurden .

Anforderungen

In dieser praktischen Übung müssen Sie eine Anwendung bereitstellen, die in der Programmiersprache Go geschrieben ist.

Eine kleine Go-Beispielanwendung ist zum Testen Ihres S2I-Builder-Images vorhanden. Die Go-Beispielanwendung stellt eine einfache HTTP-API zur Verfügung, die auf Anforderungen basierend auf der in der HTTP-Anforderung genannten Ressource antwortet.

Kapitel 5 | Anpassen von Source-to-Image-Builds

Um die praktische Übung abzuschließen, erstellen Sie ein S2I-Builder-Image für Go-basierte Anwendungen, stellen anschließend die Go-Beispielanwendung auf dem RHOC-P-Kursraumcluster bereit und testen sie gemäß den folgenden Anforderungen:

- Das S2I-Builder-Image muss den Namen `s2i-do288-go` erhalten. Das Builder-Image muss über Ihr persönliches Quay.io-Benutzerkonto verfügbar sein unter:
`quay.io/youruser/s2i-do288-go`
- Der Image-Stream für das S2I-Builder-Image erhält den Namen `s2i-do288-go`.
- Der Anwendungsname für RHOC-P lautet `greet`.
- Sowohl der Image-Stream als auch die Anwendung müssen in einem Projekt mit dem Namen `youruser-custom-s2i` erstellt werden.
- Der Zugriff auf die HTTP-API für die Anwendung muss unter der folgenden URL möglich sein:

`http://greet-youruser-custom-s2i.apps.cluster.domain.example.com`

- Der Go-Anwendungsquellcode befindet sich im Git-Repository `D0288-apps` im Verzeichnis `go-hello`.
- Bevor Sie das Builder-Image an Ihr Quay.io-Benutzerkonto übertragen, testen Sie die Anwendung lokal auf der `workstation`-VM. Beachten Sie Folgendes beim Testen des Builder-Images:
 - Der Quellcode der Anwendung befindet sich im Verzeichnis `~/D0288/labs/custom-s2i/test/test-app`.
 - Geben Sie dem Container-Image der Testanwendung den Namen `s2i-go-app`.
 - Geben Sie dem Testcontainer den Namen `go-test`.
 - Verwenden Sie beim Testen des Containers eine zufällige Benutzer-ID, wie `1234`, um eine Ausführung auf einem RHOC-P-Cluster zu simulieren.
 - Binden Sie den Container-Port `8080` an den lokalen Port `8080`.
 - Die Anwendung gibt eine Begrüßung basierend auf der URL zurück, die die Anforderung gestellt hat. Beispiel:
`http://localhost:8080/user1` gibt die folgende Antwort zurück:

Hello user1!. Welcome!

- Löschen Sie nach Abschluss des Tests den Container `go-test`, bevor Sie mit dem nächsten Schritt fortfahren.
- Testen Sie das Erstellen der Anwendung `go-hello` aus dem Quellcode mit Ihrem `s2i-do288-go`-Builder-Image.
- Nachdem Sie die Anwendung `go-hello` getestet haben, die auf RHOC-P ausgeführt wird, passen Sie das `run`-Skript für das Builder-Image `s2i-do288-go` an, indem Sie es im Quellcode der Anwendung `go-hello` überschreiben.
- Übergeben und übertragen Sie Ihr benutzerdefiniertes `run`-Skript an Ihr GitHub-Repository, das als Quelle für den Anwendungs-Build verwendet wird. Starten Sie dann

einen neuen Build und überprüfen Sie, ob die neue Version der Anwendung go-hello die Begrüßung auf Spanisch zurückgibt, z. B.:

```
Hola user1!. Bienvenido!
```

Anweisungen

1. Die S2I-Skripts für das Builder-Image sind im Ordner /home/student/D0288/labs/custom-s2i/s2i/bin enthalten. Überprüfen Sie die Skripts assemble, run und usage.
2. Bearbeiten Sie das Dockerfile für das Builder-Image und fügen Sie eine Anweisung hinzu, mit der die S2I-Skripts an die passende Stelle im Builder-Image kopiert werden. Fügen Sie diese neue Anweisung unmittelbar nach dem TODO-Kommentar hinzu, der bereits in der Datei vorhanden ist.
3. Erstellen Sie das S2I-Builder-Image. Geben Sie dem Image den Namen s2i-do288-go.
4. Erstellen Sie lokal auf der workstation-VM im Verzeichnis /home/student/D0288/labs/custom-s2i/test/test-app ein Dockerfile für ein Anwendungscontainer-Image, das das S2I-Builder-Image und den Anwendungsquellcode kombiniert.
5. Übertragen Sie das S2I-Builder-Image s2i-do288-go an Ihr persönliches Quay.io-Benutzerkonto.
6. Erstellen Sie einen Image-Stream mit dem Namen s2i-do288-go für das S2I-Builder-Image s2i-do288-go. Erstellen Sie den Image-Stream in einem Projekt mit dem Namen youruser-custom-s2i.
7. Wechseln Sie zu Ihrem lokalen Klon des D0288-apps-Git-Repository und checken Sie den master-Branch des Kurs-Repository aus, um sicherzustellen, dass Sie diese Übung von einem als fehlerfrei bekannten Zustand aus starten:
8. Erstellen Sie einen neuen Branch, in dem Sie alle Änderungen speichern können, die Sie während dieser Übung vornehmen, und übertragen Sie ihn zu GitHub:
9. Stellen Sie die Anwendung go-hello aus Ihrem persönlichen GitHub-Fork des D0288-apps-Repository auf dem RHOC-P-Kursraumcluster bereit, und testen Sie sie. Achten Sie darauf, dass Sie beim Bereitstellen der Anwendung auf den Branch custom-s2i verweisen, den Sie im vorherigen Schritt erstellt haben. Die Anwendung sendet eine Begrüßung an die in der HTTP-Anforderung angegebene Ressource zurück.

Beispiel: Beim Aufrufen der Anwendung mit der folgenden URL `http://greet-youruser-custom-s2i.apps.cluster.domain.example.com/user1` wird die folgende Antwort zurückgegeben:

```
Hello user1!. Welcome!
```

10. Passen Sie das Skript run für das Builder-Image s2i-do288-go in der Anwendungsquelle an. Ändern Sie das Startverhalten der Anwendung, indem Sie im Startabschnitt ein --lang es-Argument hinzufügen. Dadurch wird die von der Anwendung verwendete Standardsprache geändert.
Übergeben und übertragen Sie Ihre Änderungen an den Branch, den Sie als Eingabequelle für den Anwendungs-Build verwendet haben.
11. Erstellen Sie die Anwendung neu und testen Sie sie. Die Anwendung sollte nun auf Anforderungen in Spanisch reagieren.

Kapitel 5 | Anpassen von Source-to-Image-Builds

Beispiel: Beim Aufrufen der Anwendung mit der folgenden URL:

`http://greet-youruser-custom-s2i.apps.cluster.domain.example.com/`
user1 wird die folgende Antwort zurückgegeben:

```
Hola user1!. Bienvenido!
```

12. Bewerten Sie Ihre Arbeit.

Führen Sie den folgenden Befehl auf der workstation-VM aus, um zu überprüfen, ob alle Aufgaben abgeschlossen wurden:

```
[student@workstation ~]$ lab custom-s2i grade
```

13. Führen Sie eine Bereinigung durch. Führen Sie die folgenden Schritte aus:

Beenden

Führen Sie auf der workstation-VM den Befehl `lab custom-s2i finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken.

```
[student@workstation ~]$ lab custom-s2i finish
```

Hiermit ist die praktische Übung abgeschlossen.

► Lösung

Anpassen von Source-to-Image-Builds

In dieser praktischen Übung erstellen Sie ein S2I-Builder-Image zur Ausführung von Anwendungen basierend auf der Programmiersprache „Go“.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Erstellen eines S2I-Builder-Images mit der Programmiersprache „Go“, basierend auf dem Universal Base Image für RHEL 8
- Lokales Testen des S2I-Builder-Images durch Erstellen eines Dockerfiles mit dem Tool `s2i` und anschließendem Erzeugen und Ausführen des resultierenden Container-Images mit Podman
- Veröffentlichen des Builder-Images in Ihrem persönlichen Quay.io-Benutzerkonto
- Verwenden des S2I-Builder-Images für Bereitstellung und Test einer Anwendung auf einem Red Hat OpenShift Container Platform(RHOCOP)-Cluster
- Anpassen des Skripts `run` zum Ändern des Verhaltens der Anwendung und erneutem Bereitstellen der Anwendung zum Testen der Änderungen

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um dieses Lab durchzuführen zu können:

- Auf einen aktiven RHOCOP-Cluster
- Auf das Befehlszeilentool `s2i`.
- Auf einen Fork im Git-Repository, der den Quellcode der Anwendung `go-hello` enthält

Führen Sie den folgenden Befehl auf der `workstation`-VM aus, um die Voraussetzungen zu überprüfen. Dieser Befehl lädt zudem die Hilfs- und Lösungsdateien für die Wiederholungsübung herunter:

```
[student@workstation ~]$ lab custom-s2i start
```

Das Setup-Skript erstellt ein Verzeichnis namens `custom-s2i` im Verzeichnis `/home/student/D0288/labs`. Dieser Ordner enthält die Vorlagenordnerstruktur und die Dateien, die mit dem Befehl `s2i create` erstellt wurden .

Anforderungen

In dieser praktischen Übung müssen Sie eine Anwendung bereitstellen, die in der Programmiersprache Go geschrieben ist.

Eine kleine Go-Beispielanwendung ist zum Testen Ihres S2I-Builder-Images vorhanden. Die Go-Beispielanwendung stellt eine einfache HTTP-API zur Verfügung, die auf Anforderungen basierend auf der in der HTTP-Anforderung genannten Ressource antwortet.

Kapitel 5 | Anpassen von Source-to-Image-Builds

Um die praktische Übung abzuschließen, erstellen Sie ein S2I-Builder-Image für Go-basierte Anwendungen, stellen anschließend die Go-Beispielanwendung auf dem RHOC-P-Kursraumcluster bereit und testen sie gemäß den folgenden Anforderungen:

- Das S2I-Builder-Image muss den Namen `s2i-do288-go` erhalten. Das Builder-Image muss über Ihr persönliches Quay.io-Benutzerkonto verfügbar sein unter:
`quay.io/youruser/s2i-do288-go`
- Der Image-Stream für das S2I-Builder-Image erhält den Namen `s2i-do288-go`.
- Der Anwendungsname für RHOC-P lautet `greet`.
- Sowohl der Image-Stream als auch die Anwendung müssen in einem Projekt mit dem Namen `youruser-custom-s2i` erstellt werden.
- Der Zugriff auf die HTTP-API für die Anwendung muss unter der folgenden URL möglich sein:

`http://greet-youruser-custom-s2i.apps.cluster.domain.example.com`

- Der Go-Anwendungsquellcode befindet sich im Git-Repository `D0288-apps` im Verzeichnis `go-hello`.
- Bevor Sie das Builder-Image an Ihr Quay.io-Benutzerkonto übertragen, testen Sie die Anwendung lokal auf der `workstation`-VM. Beachten Sie Folgendes beim Testen des Builder-Images:
 - Der Quellcode der Anwendung befindet sich im Verzeichnis `~/D0288/labs/custom-s2i/test/test-app`.
 - Geben Sie dem Container-Image der Testanwendung den Namen `s2i-go-app`.
 - Geben Sie dem Testcontainer den Namen `go-test`.
 - Verwenden Sie beim Testen des Containers eine zufällige Benutzer-ID, wie `1234`, um eine Ausführung auf einem RHOC-P-Cluster zu simulieren.
 - Binden Sie den Container-Port `8080` an den lokalen Port `8080`.
 - Die Anwendung gibt eine Begrüßung basierend auf der URL zurück, die die Anforderung gestellt hat. Beispiel:
`http://localhost:8080/user1` gibt die folgende Antwort zurück:

Hello user1!. Welcome!

- Löschen Sie nach Abschluss des Tests den Container `go-test`, bevor Sie mit dem nächsten Schritt fortfahren.
- Testen Sie das Erstellen der Anwendung `go-hello` aus dem Quellcode mit Ihrem `s2i-do288-go`-Builder-Image.
- Nachdem Sie die Anwendung `go-hello` getestet haben, die auf RHOC-P ausgeführt wird, passen Sie das `run`-Skript für das Builder-Image `s2i-do288-go` an, indem Sie es im Quellcode der Anwendung `go-hello` überschreiben.
- Übergeben und übertragen Sie Ihr benutzerdefiniertes `run`-Skript an Ihr GitHub-Repository, das als Quelle für den Anwendungs-Build verwendet wird. Starten Sie dann

einen neuen Build und überprüfen Sie, ob die neue Version der Anwendung go-hello die Begrüßung auf Spanisch zurückgibt, z. B.:

```
Hola user1!. Bienvenido!
```

Anweisungen

1. Die S2I-Skripts für das Builder-Image sind im Ordner `/home/student/D0288/labs/custom-s2i/s2i/bin` enthalten. Überprüfen Sie die Skripts `assemble`, `run` und `usage`.
2. Bearbeiten Sie das Dockerfile für das Builder-Image und fügen Sie eine Anweisung hinzu, mit der die S2I-Skripts an die passende Stelle im Builder-Image kopiert werden. Fügen Sie diese neue Anweisung unmittelbar nach dem `TODO`-Kommentar hinzu, der bereits in der Datei vorhanden ist.
 - 2.1. Bearbeiten Sie das Dockerfile unter `~/D0288/labs/custom-s2i/Dockerfile`, und fügen Sie die folgende `COPY`-Anweisung nach dem `TODO`-Kommentar hinzu. Sie können auch die Anweisung aus der unter `~/D0288/solutions/custom-s2i/Dockerfile` bereitgestellten Dockerfile-Lösungsdatei kopieren:

```
COPY ./s2i/bin/ /usr/libexec/s2i
```

3. Erstellen Sie das S2I-Builder-Image. Geben Sie dem Image den Namen `s2i-do288-go`.
 - 3.1. Erstellen Sie das S2I-Builder-Image:

```
[student@workstation ~]$ cd ~/D0288/labs/custom-s2i  
[student@workstation custom-s2i]$ podman build -t s2i-do288-go .  
STEP 1: FROM registry.access.redhat.com/ubi8/ubi:8.0  
Getting image source signatures  
...output omitted...  
Installed:  
 golang-1.12.8-2.module+el8.1.0+4089+be929cf8.x86_64  
...output omitted...  
STEP 23: COMMIT s2i-do288-go
```

- 3.2. Überprüfen Sie, ob das Builder-Image erstellt wurde:

```
[student@workstation custom-s2i]$ podman images  
REPOSITORY TAG IMAGE ID ...  
localhost/s2i-do288-go latest d1f856d10fa7 ...  
...output omitted...
```

4. Erstellen Sie lokal auf der `workstation`-VM im Verzeichnis `/home/student/D0288/labs/custom-s2i/test/test-app` ein Dockerfile für ein Anwendungscontainer-Image, das das S2I-Builder-Image und den Anwendungsquellcode kombiniert.
 - 4.1. Erstellen Sie ein Verzeichnis mit dem Namen `s2i-go-app`, in dem der Befehl `s2i` das Dockerfile speichern kann.

```
[student@workstation custom-s2i]$ mkdir /home/student/s2i-go-app
```

Kapitel 5 | Anpassen von Source-to-Image-Builds

- 4.2. Erstellen Sie mit dem Befehl `s2i build` ein Dockerfile für das Anwendungscontainer-Image:

```
[student@workstation custom-s2i]$ s2i build test/test-app/ \
s2i-do288-go s2i-go-app \
--as-dockerfile /home/student/s2i-go-app/Dockerfile
Application dockerfile generated in /home/student/s2i-go-app/Dockerfile
```

- 4.3. Erstellen Sie anhand des generierten Dockerfiles ein Testcontainer-Image.

```
[student@workstation custom-s2i]$ cd ~/s2i-go-app
[student@workstation s2i-go-app]$ podman build -t s2i-go-app .
STEP 1: FROM s2i-do288-go
STEP 2: LABEL "io.k8s.display-name"="s2i-go-app"
...output omitted...
STEP 15: COMMIT s2i-go-app
```

- 4.4. Überprüfen Sie, ob das Anwendungscontainer-Image erstellt wurde:

```
[student@workstation s2i-go-app]$ podman images
REPOSITORY                      TAG      IMAGE ID      ...
localhost/s2i-go-app            latest   7d3d8f894f2f   ...
...output omitted...
```

- 4.5. Testen Sie das Anwendungscontainer-Image lokal auf der workstation-VM.

Führen Sie den Container mit dem Flag `-u` aus, um eine Ausführung durch einen zufälligen Benutzer auf einem RHOC-Cluster zu simulieren. Sie können den Befehl kopieren oder direkt aus dem Skript `~/D0288/labs/custom-s2i/local-test.sh` ausführen:

```
[student@workstation s2i-go-app]$ podman run --name go-test -u 1234 \
-p 8080:8080 -d s2i-go-app
18b903cfaae4...
```

- 4.6. Überprüfen Sie, ob der Container gestartet wurde:

```
[student@workstation s2i-go-app]$ podman ps
CONTAINER ID  IMAGE                      COMMAND
18b903cfaae4  localhost/s2i-go-app:latest  /bin/sh -c /usr/lib...
```

- 4.7. Führen Sie den Befehl `curl` aus, um die Anwendung zu testen:

```
[student@workstation s2i-go-app]$ curl http://localhost:8080/user1
Hello user1!. Welcome!
```

- 4.8. Beenden Sie den go-test-Anwendungscontainer:

```
[student@workstation s2i-go-app]$ podman stop go-test
18b903cfaae4...
[student@workstation s2i-go-app]$ cd ~
```

5. Übertragen Sie das S2I-Builder-Image `s2i-do288-go` an Ihr persönliches Quay.io-Benutzerkonto.

- 5.1. Laden Sie die Konfiguration Ihrer Kursumgebung.

Führen Sie den folgenden Befehl aus, um die in der ersten angeleiteten Übung erstellten Umgebungsvariablen zu laden:

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

- 5.2. Melden Sie sich mit dem Befehl `podman login` bei Ihrem Quay.io-Benutzerkonto an. Geben Sie Ihr Quay.io-Passwort ein, wenn Sie dazu aufgefordert werden.

```
[student@workstation ~]$ podman login \
-u ${RHT_OCP4_QUAY_USER} quay.io
Password:
Login Succeeded!
```

- 5.3. Veröffentlichen Sie mit dem Befehl `skopeo copy` das S2I-Builder-Image in Ihrem Quay.io-Benutzerkonto.

```
[student@workstation ~]$ skopeo copy \
containers-storage:localhost/s2i-do288-go \
docker://quay.io/${RHT_OCP4_QUAY_USER}/s2i-do288-go
...output omitted...
Writing manifest to image destination
Storing signatures
```

6. Erstellen Sie einen Image-Stream mit dem Namen `s2i-do288-go` für das S2I-Builder-Image `s2i-do288-go`. Erstellen Sie den Image-Stream in einem Projekt mit dem Namen `youruser-custom-s2i`.

- 6.1. Melden Sie sich bei RHOCUP an, und erstellen Sie ein neues Projekt. Stellen Sie dem Projektnamen Ihren Entwicklerbenutzernamen voran.

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-custom-s2i
Now using project "youruser-custom-s2i" on server "https://
api.cluster.domain.example.com:6443".
```

- 6.2. Wenn Sie noch nicht angemeldet sind, melden Sie sich mit Podman bei Ihrem persönlichen Quay.io-Benutzerkonto an, damit Sie die Datei `auth.json` exportieren können, um sie im nächsten Schritt in einem Pull Secret zu verwenden.

```
[student@workstation ~]$ podman login -u ${RHT_OCP4_QUAY_USER} quay.io
Password:
Login Succeeded!
```

- 6.3. Erstellen Sie ein Secret aus dem Zugriffstoken der Container-Registry-API, das von Podman gespeichert wurde.

Kapitel 5 | Anpassen von Source-to-Image-Builds

Sie können auch den folgenden `oc create secret`-Befehl aus dem Skript `create-secret.sh` ausführen oder ausschneiden und in das Verzeichnis `/home/student/D0288/labs/custom-s2i` einfügen.

```
[student@workstation ~]$ oc create secret generic quayio \
--from-file .dockerconfigjson=${XDG_RUNTIME_DIR}/containers/auth.json \
--type=kubernetes.io/dockerconfigjson
secret/quayio created
```

- 6.4. Verknüpfen Sie das neue Secret mit dem Servicekonto `builder`.

```
[student@workstation ~]$ oc secrets link builder quayio
```

- 6.5. Erstellen Sie einen Image-Stream, indem Sie das S2I-Builder-Image aus der privaten Kursraum-Registry importieren:

```
[student@workstation ~]$ oc import-image s2i-do288-go \
--from quay.io/${RHT_OCP4_QUAY_USER}/s2i-do288-go \
--confirm
imagestream.image.openshift.io/s2i-do288-go imported
...output omitted...
```

- 6.6. Überprüfen Sie, ob der Image-Stream erstellt wurde:

```
[student@workstation ~]$ oc get is
NAME           IMAGE REPOSITORY          ...output omitted...
s2i-do288-go   ...youruser-custom-s2i/s2i-do288-go ...output omitted...
```

7. Wechseln Sie zu Ihrem lokalen Klon des D0288-apps-Git-Repository und checken Sie den `master`-Branch des Kurs-Repository aus, um sicherzustellen, dass Sie diese Übung von einem als fehlerfrei bekannten Zustand aus starten:

```
[student@workstation ~]$ cd D0288-apps
[student@workstation D0288-apps]$ git checkout main
...output omitted...
```

8. Erstellen Sie einen neuen Branch, in dem Sie alle Änderungen speichern können, die Sie während dieser Übung vornehmen, und übertragen Sie ihn zu GitHub:

```
[student@workstation D0288-apps]$ git checkout -b custom-s2i
Switched to a new branch 'custom-s2i'
[student@workstation D0288-apps]$ git push -u origin custom-s2i
...output omitted...
 * [new branch]      custom-s2i -> custom-s2i
Branch custom-s2i set up to track remote branch custom-s2i from origin.
[student@workstation D0288-apps]$ cd ~
```

9. Stellen Sie die Anwendung `go-hello` aus Ihrem persönlichen GitHub-Fork des D0288-apps-Repository auf dem RHOC-P-Kursraumcluster bereit, und testen Sie sie. Achten Sie darauf, dass Sie beim Bereitstellen der Anwendung auf den Branch `custom-s2i` verweisen, den Sie im vorherigen Schritt erstellt haben. Die Anwendung sendet eine Begrüßung an die in der HTTP-Anforderung angegebene Ressource zurück.

Beispiel: Beim Aufrufen der Anwendung mit der folgenden URL `http://greet-youruser-custom-s2i.apps.cluster.domain.example.com/user1` wird die folgende Antwort zurückgegeben:

```
Hello user1!. Welcome!
```

- 9.1. Erstellen Sie eine neue Anwendung anhand von Quellcode in GitHub und dem Image-Stream `s2i-do288-go`:

```
[student@workstation ~]$ oc new-app --name greet \
s2i-do288-go-https://github.com/${RHT_OCP4_GITHUB_USER}/D0288-apps#custom-s2i \
--context-dir=go-hello
--> Found image a29d3e7 (About an hour old) in image stream "youruser-custom-s2i/
s2i-do288-go" under tag "latest" for "s2i-do288-go"

  Go programming language S2I builder image for D0288
...output omitted...
--> Creating resources ...
  imagestream.image.openshift.io "greet" created
  buildconfig.build.openshift.io "greet" created
  deployment.apps "greet" created
  service "greet" created
--> Success
...output omitted...
```

- 9.2. Sehen Sie sich die Build-Logs an. Warten Sie, bis der Build abgeschlossen ist und das Anwendungscontainer-Image an die RHOC-P-Registry übertragen wurde:

```
[student@workstation ~]$ oc logs -f bc/greet
Cloning "https://github.com/youruser/D0288-apps" ...
...output omitted...
--> Installing application source...
--> Building application from source...
...output omitted...
Push successful
```

- 9.3. Warten Sie, bis die Anwendung bereitgestellt wird. Zeigen Sie den Status des Anwendungs-Pods an. Der Anwendungs-Pod muss den Status `Running` aufweisen:

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
greet-1-build  0/1     Completed  0          53s
greet-6986b8fcb-fn4rq  1/1     Running   0          14s
```

- 9.4. Stellen Sie die Anwendung für den externen Zugriff mit einer Route bereit:

```
[student@workstation ~]$ oc expose svc greet
route.route.openshift.io/greet exposed
```

- 9.5. Rufen Sie die Routen-URL mit dem Befehl `oc get route` ab:

```
[student@workstation ~]$ oc get route/greet -o jsonpath='{.spec.host}{"\n"}'  
greet-youruser-custom-s2i.apps.cluster.domain.example.com
```

- 9.6. Testen Sie die Anwendung mit der Routen-URL, die Sie im vorherigen Schritt abgerufen haben:

```
[student@workstation ~]$ curl \  
http://greet-$RHT_OCP4_DEV_USER-custom-s2i.${RHT_OCP4_WILDCARD_DOMAIN}/user1  
Hello user1!. Welcome!
```

- 10.** Passen Sie das Skript `run` für das Builder-Image `s2i-do288-go` in der Anwendungsquelle an. Ändern Sie das Startverhalten der Anwendung, indem Sie im Startabschnitt ein `--lang es`-Argument hinzufügen. Dadurch wird die von der Anwendung verwendete Standardsprache geändert.

Übergeben und übertragen Sie Ihre Änderungen an den Branch, den Sie als Eingabequelle für den Anwendungs-Build verwendet haben.

- 10.1. Die S2I-Skripts können im Verzeichnis `.s2i/bin` der Anwendungsquelle angepasst werden, die sich im Verzeichnis `D0288-apps/go-hello` befindet. Erstellen Sie das Verzeichnis `.`

```
[student@workstation ~]$ mkdir -p ~/D0288-apps/go-hello/.s2i/bin
```

- 10.2. Kopieren Sie das Skript `run` im S2I-Builder-Image aus `~/D0288/labs/custom-s2i/s2i/bin/run`:

```
[student@workstation ~]$ cp ~/D0288/labs/custom-s2i/s2i/bin/run \  
~/D0288-apps/go-hello/.s2i/bin/
```

- 10.3. Passen Sie das Skript `run` an, und fügen Sie die Option `--lang es` zum Anwendungsstart hinzu. Sie können auch das vollständige Skript aus der Datei `~/D0288/solutions/custom-s2i/s2i/bin/run.es` kopieren:

```
...output omitted...  
echo "Starting app with lang option 'es'..."  
exec /opt/app-root/app --lang es
```

- 10.4. Übernehmen Sie die Änderungen in Git:

```
[student@workstation ~]$ cd ~/D0288-apps/go-hello  
[student@workstation go-hello]$ git add .  
[student@workstation go-hello]$ git commit -m "Customized run script"  
...output omitted...  
[student@workstation go-hello]$ git push  
...output omitted...  
[student@workstation go-hello]$ cd ~
```

- 11.** Erstellen Sie die Anwendung neu und testen Sie sie. Die Anwendung sollte nun auf Anforderungen in Spanisch reagieren.

Beispiel: Beim Aufrufen der Anwendung mit der folgenden URL:

`http://greet-youruser-custom-s2i.apps.cluster.domain.example.com/`
user1 wird die folgende Antwort zurückgegeben:

```
Hola user1!. Bienvenido!
```

11.1. Starten Sie einen neuen Build für die Anwendung:

```
[student@workstation ~]$ oc start-build greet  
build.build.openshift.io/greet-2 started
```

11.2. Beobachten Sie das Build-Log, und überprüfen Sie, ob ein neues Container-Image erstellt und an die interne RHOCP-Registry übertragen wird:

```
[student@workstation ~]$ oc logs -f bc/greet  
Cloning "https://github.com/youruser/D0288-apps" ...  
Commit: 0023a1b02342b633aad49e58a2eeba11dff33c3d (customized run script)  
...output omitted...  
Push successful
```

11.3. Warten Sie, bis der Anwendungs-Pod bereitgestellt wird. Der Pod muss den Status **Running** aufweisen: Verifizieren Sie den Status des Anwendungs-Pods:

```
[student@workstation ~]$ oc get pods  
NAME          READY   STATUS    RESTARTS   AGE  
greet-1-build  0/1     Completed  0          36m  
greet-2-build  0/1     Completed  0          50s  
greet-6986b8fcb-fn4rq  1/1     Running  0          36m  
...output omitted...
```

11.4. Testen Sie die Anwendung mit der Routen-URL, die Sie im Schritt 9.5 erhalten haben:

```
[student@workstation ~]$ curl \  
http://greet-${RHT_OCP4_DEV_USER}-custom-s2i.${RHT_OCP4_WILDCARD_DOMAIN}/user1  
Hola user1!. Bienvenido!
```

12. Bewerten Sie Ihre Arbeit.

Führen Sie den folgenden Befehl auf der workstation-VM aus, um zu überprüfen, ob alle Aufgaben abgeschlossen wurden:

```
[student@workstation ~]$ lab custom-s2i grade
```

13. Führen Sie eine Bereinigung durch. Führen Sie die folgenden Schritte aus:

13.1. Löschen Sie das Projekt youruser-custom-s2i in RHOCP.

```
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-custom-s2i
```

13.2. Löschen Sie alle Testcontainer, die zuvor zur lokalen Überprüfung der Anwendung erstellt wurden:

```
[student@workstation ~]$ podman rm go-test
```

- 13.3. Löschen Sie alle Container-Images, die in dieser praktischen Übung auf der workstation-VM erstellt wurden.

```
[student@workstation ~]$ podman rmi -f \
localhost/s2i-go-app \
localhost/s2i-do288-go \
registry.access.redhat.com/ubi8/ubi:8.0
...output omitted...
```

- 13.4. Löschen Sie das Image s2i-do288-go aus der externen Registry:

```
[student@workstation ~]$ skopeo delete \
docker://quay.io/${RHT_OCP4_QUAY_USER}/s2i-do288-go:latest
```

- 13.5. Melden Sie sich bei Quay.io mit Ihrem persönlichen kostenlosen Benutzerkonto an.

Navigieren Sie zu <http://quay.io> und klicken Sie auf **Sign In**, um Ihre Benutzeranmeldedaten einzugeben. Klicken Sie auf **Sign in to Quay Container Registry**, um sich bei Quay.io anzumelden.

- 13.6. Klicken Sie im Hauptmenü von Quay.io auf **Repositories** und suchen Sie nach **s2i-do288-go**. Das Schlosssymbol zeigt an, dass es sich um ein privates Repository handelt, das eine Authentifizierung für Abrufe und Übertragungen erfordert. Klicken Sie auf **s2i-do288-go**, um die Seite **Repository Activity** anzuzeigen.
- 13.7. Scrollen Sie auf der Seite **Repository Activity** für das s2i-do288-go-Repository nach unten und klicken Sie auf das Zahnradsymbol, um den den **Settings** -Tab anzuzeigen. Scrollen Sie nach unten und klicken Sie auf **Delete Repository**.
- 13.8. Klicken Sie im Dialogfeld **Delete** auf **Delete**, um das Löschen des s2i-do288-go-Repository zu bestätigen. Nach einigen Augenblicken werden Sie zur Seite **Repositories** zurückgeleitet. Sie können sich jetzt bei Quay.io abmelden.

Beenden

Führen Sie auf der workstation-VM den Befehl `lab custom-s2i finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken.

```
[student@workstation ~]$ lab custom-s2i finish
```

Hiermit ist die praktische Übung abgeschlossen.

Zusammenfassung

In diesem Kapitel wurden die folgenden Themen behandelt:

- Ein S2I-Builder-Image ist ein spezielles Container-Image, mit dem Entwickler Anwendungscontainer-Images erzeugen. Builder-Images enthalten grundlegende Betriebssystem-Libraries, Sprachlaufzeiten, Frameworks und Anwendungsabhängigkeiten sowie Source-to-Images-Tools und -Dienstprogramme.
- S2I-Builder-Images stellen standardmäßig S2I-Skripts bereit. Diese S2I-Skripts können im Anwendungsquellcode durch Hinzufügen von S2I-Skripts zum Verzeichnis `.s2i/bin` überschrieben werden.
- Mit dem Befehlszeilentool `s2i` können S2I-Builder-Images außerhalb von OpenShift erstellt und getestet werden.
- Verwenden Sie in RHEL 8- oder OpenShift 4-Umgebungen, in denen Docker standardmäßig nicht verfügbar ist, die Option `--as=dockerfile` für den Befehl `s2i build`. Der Befehl erzeugt dann eine Dockerfile und unterstützende Verzeichnisse, die Sie mit Podman erstellen können, um Ihr S2I-Builder-Image zu testen.

Kapitel 6

Bereitstellen von Anwendungen mit mehreren Containern

Ziel

Bereitstellung von Anwendungen mit mehreren Containern über Helm-Charts und Kustomize

Ziele

- Beschreiben der Elemente einer OpenShift-Vorlage
- Erstellen einer Anwendung mit mehreren Containern mit Helm-Charts
- Anpassen von OpenShift-Bereitstellungen

Abschnitte

- Erläutern der OpenShift-Vorlagen (und Test)
- Erstellen eines Helm-Chart (und angeleitete Übung)
- Anpassen von Bereitstellungen mit Kustomize (und angeleitete Übung)

Praktische Übung

Bereitstellen von Anwendungen mit mehreren Containern

Erläutern von OpenShift-Vorlagen

Ziele

Nach Abschluss dieses Abschnitts sollten Sie in der Lage sein, eine Liste der Red Hat OpenShift(RHOCP)-Ressourcen in eine RHOCP-Vorlage zu konvertieren.

Beschreiben einer Vorlage

Eine RHOCP-Vorlage ist eine YAML- oder JSON-Datei, die aus einer Reihe von RHOCP-Ressourcen besteht. Vorlagen definieren Parameter, die zum Anpassen der Ressourcenkonfiguration verwendet werden. RHOCP verarbeitet Vorlagen, indem Parameterverweise durch Werte ersetzt und ein benutzerdefinierter Satz an Ressourcen erstellt werden.

Eine Vorlage ist nützlich, wenn Sie eine Gruppe von Ressourcen als eine Einheit bereitstellen möchten, anstatt sie einzeln bereitzustellen. Beispiele für Anwendungsfälle für Vorlagen:

- Ein unabhängiger Softwareanbieter (Independent Software Vendor, ISV) stellt eine Vorlage für die Bereitstellung seines Produkts auf RHOCP bereit. Die Vorlage enthält Konfigurationsdetails der Container, aus denen das Produkt besteht. Zudem enthält sie eine Deploymentkonfiguration für die Anzahl der Replikate, Services und Routen, persistente Storage-Konfiguration, Health Checks und Ressourcenlimits für Computing-Ressourcen wie CPU, Arbeitsspeicher und E/A.
- Ihre mehrschichtige Anwendung besteht aus einer Reihe separater Komponenten, z. B. einem Webserver, einem Anwendungsserver und einer Datenbank. Sie sollten diese Komponenten zusammen als eine Einheit auf RHOCP bereitstellen, um die Bereitstellung der Anwendung in einer Staging-Umgebung zu vereinfachen. Auf diese Weise können Ihre QS- und Akzeptanztestteams schnell Anwendungen für Tests bereitstellen.



Anmerkung

RHOCP-Vorlagen sind veraltet und werden in zukünftigen RHOCP-Versionen entfernt. RHOCP-Vorlagen werden zwar weiterhin unterstützt, sie werden hier aber nur der Vollständigkeit halber behandelt und nicht von Red Hat empfohlen.

Vorlagensyntax

Die Syntax einer RHOCP-Vorlage folgt der allgemeinen Syntax einer RHOCP-Ressource, jedoch mit dem Attribut `objects` anstelle des Attributs `spec`. Eine Vorlage umfasst in der Regel die Attribute `parameters` und `labels` sowie Annotationen in den zugehörigen Metadaten.

In der folgenden Auflistung werden eine Beispielvorlagendefinition im YAML-Format gezeigt und die Hauptsyntaxelemente veranschaulicht. Wie bei jeder RHOCP-Ressource können Sie auch Vorlagen in der JSON-Syntax definieren:

```
apiVersion: template.openshift.io/v1
kind: Template ①
metadata:
  name: mytemplate
```

```
annotations:  
  description: "Description" ❷  
objects: ❸  
- apiVersion: v1  
  kind: Pod  
  metadata:  
    name: myapp  
  spec:  
    containers:  
      - env:  
        - name: MYAPP_CONFIGURATION  
          value: ${MYPARAMETER} ❹  
      image: myorganization/myapplication  
      name: myapp  
      ports:  
        - containerPort: 80  
          protocol: TCP  
parameters: ❺  
- description: Myapp configuration data  
  name: MYPARAMETER  
  required: true  
labels: ❻  
  mylabel: myapp
```

- ❶ Vorlagenressourcentyp
- ❷ Optionale Annotationen zur Verwendung durch RHOCP-Tools
- ❸ Ressourcenliste
- ❹ Verweis auf einen Vorlagenparameter
- ❺ Parameterliste
- ❻ Bezeichnungsliste

Die Ressourcenliste einer Vorlage enthält in der Regel weitere Ressourcen wie Build-Konfigurationen, Deploymentkonfigurationen, persistente Volume-Anforderungen (Persistent Volume Claim, PVC), Services und Routen.

Attribute in der Ressourcenliste können auf den Wert eines beliebigen Parameters verweisen.

Sie können benutzerdefinierte Bezeichnungen für eine Vorlage definieren. RHOCP fügt diese Bezeichnungen allen Ressourcen hinzu, die von der Vorlage erstellt werden.

Wenn in einer Vorlage mehrere Ressourcen definiert sind, ist es entscheidend, die Reihenfolge der Definition dieser Ressourcen zu berücksichtigen, um den Abhängigkeiten zwischen Ressourcen gerecht zu werden. Von RHOCP wird kein Fehler gemeldet, wenn eine Ressource auf eine nicht vorhandene abhängige Ressource verweist.

Ein von einer Ressource ausgelöster Prozess schlägt möglicherweise fehl, wenn er vor einer abhängigen Ressource gestartet wird. Ein Beispiel dafür ist eine Build-Konfiguration, die auf einen Image-Stream als Ausgabe-Image verweist, und Ihre Vorlage definiert diesen Image-Stream nach der Build-Konfiguration. In diesem Szenario ist es möglich, dass die Build-Konfiguration einen Build startet, bevor der Image-Stream vorhanden ist.

Definieren von Vorlagenparametern

Die vorherige Beispielvorlage enthielt die Definition eines erforderlichen Parameters. Optionale und erforderliche Parameter können Standardwerte bereitstellen. Beispiel:

```
parameters:  
- description: Myapp configuration data  
  name: MYPARAMETER  
  value: /etc/myapp/config.ini
```

RHOCP kann zufällige Standardwerte für Parameter generieren. Dies ist nützlich für Secrets und Passwörter:

```
parameters:  
- description: ACME cloud provider API key  
  name: APIKEY  
  generate: expression  
  from:"[a-zA-Z0-9]{12}"
```

Die Syntax für generierte Werte ist eine Teilmenge der regulären Perl-Ausdruckssyntax. Die vollständige Vorlagenparameter-Ausdruckssyntax finden Sie in den Referenzen am Ende dieses Abschnitts.

Hinzufügen einer Vorlage zu OpenShift

Fügen Sie mit dem Befehl `oc create` RHOCP eine Vorlage hinzu oder verwenden Sie die Web Console (mit der Seite für den Import von YAML), um die Vorlagenressource wie bei jedem anderen Ressourcentyp anhand der Vorlagendefinitionsdatei zu erstellen.

Es ist eine gängige Praxis, Projekte zu erstellen, die ausschließlich Vorlagen enthalten, da RHOCP es ermöglicht, dass Vorlagen von mehreren Benutzern und Projekten gemeinsam genutzt werden. Diese Projekte enthalten in der Regel andere potenziell gemeinsam genutzte Ressourcen wie Image-Streams.

Eine Standardinstallation von Red Hat RHOCP Container Platform enthält im Projekt `openshift` mehrere Vorlagen. Alle RHOCP-Cluster-Benutzer haben Lesezugriff auf das Projekt `openshift`. Nur Cluster-Administratoren besitzen aber die Berechtigung, Vorlagen in diesem Projekt zu erstellen oder zu löschen.

Erstellen einer Anwendung anhand einer Vorlage

Sie können eine Anwendung direkt aus einer Vorlagendatei für die Ressourcendefinition bereitstellen. Der Befehl `oc new-app` und der Befehl `oc process` verwenden die Vorlagendatei als Eingabe und verarbeiten sie, um Parameter anzuwenden und Ressourcen zu erstellen.

Sie können auch eine Vorlagendatei an den Befehl `oc create` übergeben, um eine Vorlagenressource in RHOCP zu erstellen. Wenn die Vorlage von mehreren Entwicklern wiederverwendet werden soll, empfiehlt es sich, die Vorlagenressource in einem freigegebenen Projekt zu erstellen. Wenn die Vorlage von einer einzelnen Bereitstellung verwendet werden soll, sollte sie in einer Datei beibehalten werden.

Der Befehl `oc new-app` erstellt Ressourcen anhand der Vorlage. Demgegenüber erstellt der Befehl `oc process` eine Ressourcenliste anhand der Vorlage. Sie müssen die durch den Befehl `oc process` generierte Ressourcenliste in einer Datei speichern oder als Eingabe an den Befehl `oc create` übergeben, um die Ressourcen aus der Liste zu erstellen.

Kapitel 6 | Bereitstellen von Anwendungen mit mehreren Containern

Sowohl für den Befehl `oc new-app` als auch den Befehl `oc process` muss jeder Parameterwert mithilfe einer anderen -p-Option angegeben werden. Eine weitere Option, die von beiden Befehlen akzeptiert wird, ist die Option -l. Diese fügt allen anhand der Vorlage erstellten Ressourcen eine Bezeichnung hinzu.

Im Folgenden finden Sie einen `oc new-app`-Beispielbefehl, der eine Anwendung anhand einer Vorlagendatei bereitstellt:

```
[user@host ~]$ oc new-app --file mytemplate.yaml -p PARAM1=value1 \
-p PARAM2=value2
```

Im Folgenden finden Sie einen `oc process`-Beispielbefehl, der Werte auf eine Vorlage anwendet und die Ergebnisse in einer lokalen Datei speichert:

```
[user@host ~]$ oc process -f mytemplate.yaml -p PARAM1=value1 \
-p PARAM2=value2 > myresourcelist.yaml
```

Die vom vorherigen Beispiel generierte Datei wird dann an den Befehl `oc create` weitergegeben:

```
[user@host ~]$ oc create -f myresourcelist.yaml
```

Sie können die vorherigen zwei Beispiele mit einer Pipe kombinieren:

```
[user@host ~]$ oc process -f mytemplate.yaml -p PARAM1=value1 \
-p PARAM2=value2 | oc create -f -
```

Red Hat empfiehlt die Verwendung des Befehls `oc new-app` anstelle des Befehls `oc process`. Sie können den Befehl `oc process` mit der Option -f verwenden, um nur die Parameter aufzulisten, die von der angegebenen Vorlage definiert wurden:

```
[user@host ~]$ oc process -f mytemplate.yaml --parameters
```



Anmerkung

Es gibt keinen Abschnitt in der RHOC 4.6 Web Console zum Auflisten oder Anzeigen von Vorlagen. Sie können jedoch die verfügbaren Vorlagen auf den Seiten Administrator → Home → Search und Developer → Search auflisten.

Standardmäßig können Sie in der RHOC 4.6 Web Console keine Anwendungen mit benutzerdefinierten Vorlagen erstellen. Cluster-Administratoren können zusätzliche Vorlagen im Developer Catalog installieren. Vorlagen im Developer Catalog können zum Erstellen neuer Anwendungen verwendet werden.



Literaturhinweise

Weitere Informationen finden Sie im Kapitel *Using Templates* der Dokumentation für Red Hat OpenShift Container Platform 4.6 unter
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/images/using-templates#using-templates

► Quiz

Erläutern von OpenShift-Vorlagen

Wählen Sie die richtigen Antworten auf die folgenden Fragen aus:

► 1. Welchen Zweck erfüllt eine OpenShift-Vorlage?

- a. Beschreiben Sie die Ressourcenkonfiguration für einen einzelnen Container.
- b. Kapselt einen Satz von OpenShift-Ressourcen für die Wiederverwendung.
- c. Stellt benutzerdefinierte Konfigurationen für einen OpenShift-Cluster bereit.
- d. Passt das Erscheinungsbild der Web Console an.

► 2. Welche zwei der folgenden Ansätze definiert einen Vorlagenparameter als optional? (Wählen Sie zwei Antworten aus.)

- a. Das Festlegen des Attributs required auf false.
- b. Das Festlegen des Attributs required auf not.
- c. Das Weglassen des Attributs required.
- d. Das Festlegen des Attributs required auf expression.
- e. Das Festlegen des Attributs from auf einen regulären Ausdruck.

► 3. Welcher der folgenden Befehle wird zum Erstellen von Ressourcen aus einer Vorlage empfohlen?

- a. oc export
- b. oc new-app
- c. oc create
- d. oc process
- e. Keiner der aufgeführten Befehle

► 4. Welche drei Aussagen beschreiben gültige Speicherorte für das Einfügen von Vorlagenparameterverweisen? (Wählen Sie drei Antworten aus.)

- a. Als Name einer Umgebungsvariable in einer Pod-Ressource.
- b. Als Schlüssel für eine Bezeichnung in allen durch die Vorlage erstellten Ressourcen.
- c. Als Wert für das Attribut host in einer Routenressource.
- d. Als Wert für eine Anmerkung in der Vorlage.
- e. Als Wert einer Umgebungsvariable in einer Pod-Ressource.

► Lösung

Erläutern von OpenShift-Vorlagen

Wählen Sie die richtigen Antworten auf die folgenden Fragen aus:

► 1. Welchen Zweck erfüllt eine OpenShift-Vorlage?

- a. Beschreiben Sie die Ressourcenkonfiguration für einen einzelnen Container.
- b. Kapselt einen Satz von OpenShift-Ressourcen für die Wiederverwendung.
- c. Stellt benutzerdefinierte Konfigurationen für einen OpenShift-Cluster bereit.
- d. Passt das Erscheinungsbild der Web Console an.

► 2. Welche zwei der folgenden Ansätze definiert einen Vorlagenparameter als optional? (Wählen Sie zwei Antworten aus.)

- a. Das Festlegen des Attributs `required` auf `false`.
- b. Das Festlegen des Attributs `required` auf `not`.
- c. Das Weglassen des Attributs `required`.
- d. Das Festlegen des Attributs `required` auf `expression`.
- e. Das Festlegen des Attributs `from` auf einen regulären Ausdruck.

► 3. Welcher der folgenden Befehle wird zum Erstellen von Ressourcen aus einer Vorlage empfohlen?

- a. `oc export`
- b. `oc new-app`
- c. `oc create`
- d. `oc process`
- e. Keiner der aufgeführten Befehle

► 4. Welche drei Aussagen beschreiben gültige Speicherorte für das Einfügen von Vorlagenparameterverweisen? (Wählen Sie drei Antworten aus.)

- a. Als Name einer Umgebungsvariable in einer Pod-Ressource.
- b. Als Schlüssel für eine Bezeichnung in allen durch die Vorlage erstellten Ressourcen.
- c. Als Wert für das Attribut `host` in einer Routenressource.
- d. Als Wert für eine Anmerkung in der Vorlage.
- e. Als Wert einer Umgebungsvariable in einer Pod-Ressource.

Erstellen eines Helm-Chart

Helm-Charts

Helm ist ein Open Source-Paket-Manager für Kubernetes-Anwendungen. Damit kann der Lifecycle dieser Kubernetes-Anwendungen gepackt, gemeinsam verwendet und verwaltet werden.

Ein Helm-Chart ist eine Sammlung von Dateien und Vorlagen, die eine Helm-Anwendung definieren. Diese Dateien werden später für die Verteilung mit Helm-Repositories gepackt.

Helm-Lifecycle

Der Helm-CLI-Befehl verwaltet den Lifecycle von Helm-Charts anhand der folgenden grundlegenden Befehle:

Helm-Befehle

Befehl	Beschreibung
dependency	Verwalten der Abhängigkeiten eines Diagramms
install	Installieren eines Diagramms
list	Auflisten installierter Releases
pull	Herunterladen eines Diagramms aus einem Repository
rollback	Rollback einer Version auf eine frühere Revision
search	Suche nach einem Keyword in Diagrammen
show	Anzeigen von Informationen eines Diagramms
status	Anzeigen des Status der benannten Version
uninstall	Deinstallieren einer Version
upgrade	Upgrade eines Release

Helm-Charts-Struktur

Mit dem Befehl `helm create` werden die benötigten Dateien in der richtigen Struktur erstellt. Die von diesem Befehl erstellten Dateien sind die grundlegenden Dateien, die für ein Helm-Chart benötigt werden. Sie enthalten die Minimalvorlagen, die erforderlich sind, damit eine Anwendung funktionsfähig ist.

Ein Helm-Chart besteht hauptsächlich aus zwei YAML-Dateien und einer Liste von Vorlagen.

Kapitel 6 | Bereitstellen von Anwendungen mit mehreren Containern

Die YAML-Dateien sind:

- **Chart.yaml**: Enthält die Informationen zur Diagrammdefinition.
- **values.yaml**: Enthält die Werte, die Helm in den Standardvorlagen und in den vom Benutzer erstellten Vorlagen verwendet.

Zusätzlich zu diesen beiden Dateien enthält ein Helm-Chart mehrere Vorlagendateien. Diese Dateien bilden die Grundlage für die Kubernetes-Ressourcen, aus denen die Anwendung besteht.

Die Grundstruktur der Datei **Chart.yaml** sieht folgendermaßen aus:

```
apiVersion: v2 ①
name: mychart ②
description: A Helm chart for Kubernetes ③
type: application ④
version: 0.1.0 ⑤
appVersion: "1.0.0" ⑥
```

- ① Version der zu verwendenden Helm-API
- ② Name des Helm-Charts
- ③ Beschreibung des Helm-Charts
- ④ Typ des Helm-Charts: Anwendung oder Library
- ⑤ Version des Helm-Charts
- ⑥ Version der Anwendung dieses Diagrammpakets

Die Datei **Chart.yaml** kann andere optionale Werte enthalten. Einer der wichtigsten Werte ist der Abschnitt **dependencies**. Der Abschnitt **dependencies** enthält eine Liste anderer Helm-Charts, welche ermöglichen, dass dieses Helm Chart funktioniert.

Die Struktur des Abschnitts **dependencies** sieht folgendermaßen aus:

```
dependencies: ①
  - name: dependency1 ②
    version: 1.2.3 ③
    repository: https://examplerrepo.com/charts ④
  - name: dependency2
    version: 3.2.1
    repository: https://helmrepo.example.com/charts
```

- ① Liste von Abhängigkeiten
- ② Name des ersten erforderlichen Helm-Charts
- ③ Diagrammversion, von der Abhängigkeit bestehen soll
- ④ Repository, in dem sich das abhängige Helm-Chart befindet

Diagrammwerte

Helm verarbeitet die Vorlagendateien im Diagramm und ersetzt die Platzhalter während der Verarbeitung des Diagramms durch die tatsächlichen Werte. Sie können diese Werte statisch bereitstellen, indem Sie die Datei `values.yaml` verwenden, oder sie mithilfe des Flags `--set` des Befehlszeilentools `helm` dynamisch während der Paketierung oder Installation bereitstellen.

Die gängigste Vorgehensweise besteht darin, die Mehrheit der Werte durch Verwendung der Datei `values.yaml` bereitzustellen und die dynamische Bereitstellung nur für Werte zu verwenden, die Sie zur Installationszeit bereitstellen müssen.

Im Folgenden finden Sie einen Auszug aus dem Inhalt dieser Datei:

```
...
image:
  repository: container.repo/name ①
  pullPolicy: IfNotPresent ②
  tag: "2.1" ③
...
serviceAccount:
  create: true ④
  annotations: {}
  name: "" ⑤
...
service:
  type: ClusterIP ⑥
  port: 80 ⑦
```

- ① Link zum Container-Image der Anwendung
- ② Container-Pull-Richtlinie im Kubernetes-Cluster
- ③ Zu verwendendes Container-Tag, Standardwert ist die `appVersion` des Diagramms
- ④ Ob ein neues Servicekonto für die Anwendung erstellt werden soll oder nicht
- ⑤ Name des Servicekontos, automatisch generiert, wenn leer
- ⑥ Typ des Kubernetes-Services
- ⑦ Externer Anwendungs-Port

Vorlagen

Helm verwendet Vorlagen, um zur Laufzeit die Ressourcen zu erstellen, die zur Bereitstellung der Anwendung im Kubernetes-Cluster erforderlich sind. Mit dem Helm-CLI-Tool werden einige dieser Vorlagen erstellt. Sie können sie jedoch ändern oder neue Vorlagen erstellen, die Ihren Anforderungen entsprechen.

Helm verwendet die Go-Vorlagsprache, um die Vorlagen im Verzeichnis `templates` sowie einige andere Funktionen zu definieren.

Kapitel 6 | Bereitstellen von Anwendungen mit mehreren Containern

In einem Abschnitt der allgemeinen `deployment.yaml`-Vorlagendatei können Sie die Verwendung von Platzhaltern und bedingten Abschnitten sehen:

```
metadata:  
  name: {{ include "mychart.fullname" . }} ①  
  labels:  
    {{- include "mychart.labels" . | nindent 4 }}  
spec:  
  {{- if not .Values.autoscaling.enabled }} ②  
  replicas: {{ .Values.replicaCount }} ③  
  {{- end }} ④  
  template:  
    spec:  
      containers:  
        - name: {{ .Chart.Name }} ⑤
```

- ① Das Präfix für Werte aus der Datei `Chart.yaml` ist der Name des Diagramms
- ② Den Block ausgeben, wenn der Wert „false“ ist
- ③ Das Präfix für Werte aus der Datei `values.yaml` ist `Values`
- ④ Den bedingten Block beenden
- ⑤ Das Präfix für den Namen des Diagramms ist `Chart`



Literaturhinweise

Helm

<https://helm.sh/>

Go-Vorlagentensprache

<https://golang.org/pkg/text/template/>

► Angeleitete Übung

Erstellen eines Helm-Chart

In dieser Übung erstellen Sie ein Helm-Chart für eine Anwendung mit mehreren Containern und stellen es in einem Red Hat OpenShift(RHOCP)-Cluster bereit.

Ergebnisse

Sie sollten in der Lage sein, ein Helm-Chart für die Anwendung „Famous Quotes“ und ihre Abhängigkeiten von einem RHOCP-Cluster zu erstellen.

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen konfigurierten und aktiven RHOCP-Cluster
- Auf das Helm-CLI-Tool

Führen Sie auf dem Rechner `workstation` als Benutzer `student` den Befehl `lab` aus, um die Voraussetzungen für diese Übung zu überprüfen:

```
[student@workstation ~]$ lab multicontainer-helm start
```

Anweisungen

► 1. Erstellen Sie ein neues Helm-Chart.

1.1. Erstellen Sie das Helm-Chart `famouschart`.

Erstellen Sie ein ganz neues Helm-Chart, indem Sie den Befehl `helm create` verwenden, und nennen Sie es `famouschart`.

```
[student@workstation ~]$ cd ~/DO288/labs/multicontainer-helm  
[student@workstation multicontainer-helm]$ helm create famouschart  
Creating famouschart  
[student@workstation multicontainer-helm]$ cd famouschart
```

1.2. Überprüfen Sie die Dateistruktur.

```
[student@workstation famouschart]$ tree .  
.├── charts  
├── Chart.yaml  
└── templates  
    ├── deployment.yaml  
    ├── _helpers.tpl  
    ├── hpa.yaml  
    ├── ingress.yaml  
    └── NOTES.txt  
    └── serviceaccount.yaml
```

```
|   └── service.yaml
|   └── tests
|       └── test-connection.yaml
└── values.yaml
```

► 2. Konfigurieren Sie die Anwendungsbereitstellung.

Verwenden Sie die Datei `values.yaml`, um die resultierende Bereitstellung für die Anwendung zu konfigurieren.

2.1. Konfigurieren Sie Image- und Versionswerte.

Aktualisieren Sie die Datei `values.yaml`. Legen Sie dabei die Eigenschaft `repository` des Abschnitts `image` auf das Image `quay.io/redhattraining/famous-quotes` und die `tag`-Eigenschaft desselben Abschnitts auf den Wert `2.1` fest, um die entsprechende Version der Anwendung auszuwählen.

Der entsprechende Abschnitt der Datei `values.yaml` sollte folgendermaßen aussehen:

```
image:
  repository: quay.io/redhattraining/famous-quotes
  pullPolicy: IfNotPresent
  tag: "2.1"
```

2.2. Stellen Sie sicher, dass der Container den richtigen Port verwendet, um eine Verbindung zur Anwendung herzustellen.

Ändern Sie in der Datei `templates/deployment.yaml` die Eigenschaft `containerPort` im Abschnitt `containers` so, dass der Wert `8000` verwendet wird.

Der entsprechende Abschnitt der Datei `templates/deployment.yaml` sollte folgendermaßen aussehen:

```
ports:
- name: http
  containerPort: 8000
  protocol: TCP
```

► 3. Fügen Sie die Datenbankabhängigkeit hinzu.

Die Anwendung „Famous Quotes“ verwendet eine Datenbank zum Speichern der Zitate. Daher müssen Sie neben der Anwendung eine Datenbank bereitstellen, damit sie funktioniert. Dafür müssen Sie die Abhängigkeit für das Anwendungsdiagramm bereitstellen und das Datenbankdiagramm konfigurieren.

3.1. Fügen Sie dem Anwendungsdiagramm die Abhängigkeit `mariadb` hinzu.

Fügen Sie dazu den folgenden Ausschnitt am Ende der Datei `Chart.yaml` hinzu:

```
dependencies:
- name: mariadb
  version: 9.3.11
  repository: https://charts.bitnami.com/bitnami
```

Dazu können Sie die Inhalte von `~/D0288/labs/multicontainer-helm/dependencies.yaml` an die Datei `Chart.yaml` anhängen:

```
[student@workstation famouschart]$ cat ../dependencies.yaml >> Chart.yaml
```

- 3.2. Aktualisieren Sie die Abhängigkeiten für das Diagramm.

Dadurch werden die als Abhängigkeiten hinzugefügten Diagramme heruntergeladen und deren Versionen gesperrt.

```
[student@workstation famouschart]$ helm dependency update
Getting updates for unmanaged Helm repositories...
...Successfully got an update from the "https://charts.bitnami.com/bitnami" chart
repository
Saving 1 charts
Downloading mariadb from repo https://charts.bitnami.com/bitnami
Deleting outdated charts
```

- 3.3. Richten Sie die Datenbank so ein, dass für die Authentifizierung und aus Gründen der Sicherheit benutzerdefinierte Werte verwendet werden.

Um dieselben Werte an die bereitgestellte Anwendung zu übergeben, müssen Sie die Werte steuern, anstatt zuzulassen, dass das Helm-Chart der Datenbank Werte nach dem Zufallsprinzip erstellt.

Fügen Sie am Ende der Datei `values.yaml` die folgenden Zeilen hinzu:

```
mariadb:
  auth:
    username: quotes
    password: quotespwd
    database: quotesdb
  primary:
    podSecurityContext:
      enabled: false
    containerSecurityContext:
      enabled: false
```

Dazu können Sie die Inhalte von `~/D0288/labs/multicontainer-helm/mariadb.yaml` an die Datei `values.yaml` anhängen:

```
[student@workstation famouschart]$ cat ../mariadb.yaml >> values.yaml
```

- ▶ 4. Konfigurieren Sie den Datenbankzugriff der Anwendung mithilfe von Umgebungsvariablen.

- 4.1. Die StandardDeploymentvorlage übergibt keine Umgebungsvariable an die bereitgestellten Anwendungen. Ändern Sie die Vorlage `templates/deployment.yaml`, um die in der Datei `values.yaml` definierten Umgebungsvariablen an den Container der Anwendung zu übergeben. Fügen Sie nach dem Wert `imagePullPolicy` im Abschnitt `containers` den folgenden Ausschnitt hinzu:

```
imagePullPolicy: {{ .Values.image.pullPolicy }}
env:
{{- range .Values.env -}}
- name: {{ .name }}
  value: {{ .value }}
{{- end -}}
```



Warnung

Stellen Sie beim Umgang mit YAML-Dateien sicher, dass die Einrückung korrekt ist:
 Die Zeilen `imagePullPolicy` und `env:` müssen auf derselben Ebene sein. Die
 Einträge für `name` und `value` müssen ebenfalls auf derselben Ebene und tiefer als
 der Eintrag `env:` liegen. - im Eintrag - `name` muss auf derselben Ebene oder tiefer
 als der Eintrag `env:` liegen.

- 4.2. Fügen Sie am Ende der Datei `values.yaml` die entsprechenden Umgebungsvariablen hinzu:

```
env:
- name: "QUOTES_HOSTNAME"
  value: "famousapp-mariadb"
- name: "QUOTES_DATABASE"
  value: "quotesdb"
- name: "QUOTES_USER"
  value: "quotes"
- name: "QUOTES_PASSWORD"
  value: "quotespwd"
```

Dazu können Sie die Inhalte von `~/D0288/labs/multicontainer-helm/env.yaml` an die Datei `values.yaml` anhängen:

```
[student@workstation famouschart]$ cat ../env.yaml >> values.yaml
```

- 5. Stellen Sie die Anwendung mit dem Helm-Chart bereit.

- 5.1. Erstellen Sie ein Projekt in RHOCP.

```
[student@workstation famouschart]$ source /usr/local/etc/ocp4.config
[student@workstation famouschart]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
[student@workstation famouschart]$ oc new-project \
${RHT_OCP4_DEV_USER}-multicontainer-helm
```

- 5.2. Führen Sie den Befehl `helm install` aus, um die Anwendung im RHOCP-Cluster bereitzustellen:

```
[student@workstation famouschart]$ helm install famousapp .
NAME: famousapp
LAST DEPLOYED: Thu May 20 19:12:09 2021
NAMESPACE: youruser-multicontainer-helm
STATUS: deployed
REVISION: 1
NOTES:
...output omitted...
```

Mit diesem Befehl wird eine RHOP-Bereitstellung mit dem Namen famousapp erstellt. Führen Sie den Befehl `oc get deployments` aus, um den Status dieser Bereitstellung zu verifizieren:

```
[student@workstation famouschart]$ oc get deployments
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
famousapp-famouschart   0/1     1           1           10s
```

Es kann eine Weile dauern, bis der Pod mariadb vollständig für die Anwendung verfügbar ist. Daher ist der Status der Bereitstellung nicht betriebsbereit.

Verwenden Sie mehrmals den Befehl `oc get pods`, um zu verifizieren, dass die Anwendung und die Datenbank ordnungsgemäß bereitgestellt wurden:

```
[student@workstation famouschart]$ oc get pods
NAME                           READY   STATUS    RESTARTS   AGE
famousapp-famouschart-7bff544d88-f289l   1/1     Running   3          5m
famousapp-mariadb-0            1/1     Running   0          5m
```

► 6. Testen Sie die Anwendung.

6.1. Stellen Sie den Service der Anwendung bereit:

```
[student@workstation famouschart]$ oc expose service famousapp-famouschart
route.route.openshift.io/famousapp-famouschart exposed
```

6.2. Rufen Sie den /random-Endpunkt der bereitgestellten Anwendung auf:

```
[student@workstation famouschart]$ FAMOUS_URL=$(oc get route \
-n ${RHT_OCP4_DEV_USER}-multicontainer-helm famousapp-famouschart \
-o jsonpath='{.spec.host}' '/random')
[student@workstation famouschart]$ curl $FAMOUS_URL
8: Those who can imagine anything, can create the impossible.
- Alan Turing
```

Dieser Endpunkt gibt ein zufälliges Zitat aus der Datenbank zurück. Daher sehen Sie höchstwahrscheinlich ein anderes Zitat. Führen Sie den Aufruf erneut aus, um dieses zufällige Verhalten zu verifizieren.

```
[student@workstation famouschart]$ curl $FAMOUS_URL
1: When words fail, music speaks.
- William Shakespeare
```

Beenden

Führen Sie auf der `workstation`-VM den Befehl `lab multicontainer-helm finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken. Die Aktion „Beenden“ veröffentlicht dieses Projekt und seine Ressourcen.

```
[student@workstation famouschart]$ cd ~  
[student@workstation ~]$ lab multicontainer-helm finish
```

Hiermit ist die angeleitete Übung beendet.

Anpassen einer Bereitstellung mit Kustomize

Kustomize

Kustomize ist ein Tool, mit dem Kubernetes-Ressourcen für unterschiedliche Umgebungen oder Anforderungen angepasst werden. Kustomize ist eine Lösung ohne Vorlagen, die die Wiederverwendung von Konfigurationen unterstützt und eine einfache Möglichkeit zum Patchen von Ressourcen bietet.

Der häufigste Anwendungsfall ist die Notwendigkeit, unterschiedliche Ressourcen für verschiedene Umgebungen wie Entwicklung, Staging und Produktion zu definieren. Das ist jedoch nicht die einzige Einsatzmöglichkeit. Sie müssen entscheiden, was eine Umgebung in Ihrem Anwendungsfall definiert, welche Umgebungen Sie haben und welche unterschiedlichen Konfigurationen für jede Umgebung erforderlich sind.

Kustomize trennt diese Konfigurationssätze in zwei Typen: „Base“ und „Overlays“.

Base enthält alle Konfigurationen und Ressourcen, die allen Derivaten gemeinsam sind. Overlays stellen die Unterschiede zu Base für eine bestimmte Umgebung dar. Kustomize verwendet Verzeichnisse, um diese Konfigurationssätze zu repräsentieren.

Ein Beispiel für ein Kustomize-Layout könnte folgendermaßen aussehen:

```
myapp
└── base
└── overlays
    └── production
        └── staging
```

Kustomization-Datei

Für jeden Konfigurationssatz benötigt Kustomize eine `kustomization.yaml`-Datei, die Folgendes enthalten kann:

- Einzuschließende Ressourcendateien
- Basiskonfiguration als Grundlage
- Ressourcen, die die Basiskonfiguration patchen
- Allgemeine Ressourcendefinitionen, die für alle Konfigurationssätze gelten, die von diesem Konfigurationssatz abhängen

Diese Datei muss sich im Verzeichnis des Konfigurationssatzes befinden.

Ressourcendateien

Der Abschnitt „resources“ der Datei `kustomization.yaml` ist eine Liste von Dateien, die gemeinsam alle für diese spezifische Umgebung erforderlichen Ressourcen erstellen. Beispiel:

```
resources:  
- deployment.yaml  
- secrets.yaml  
- service.yaml
```

Das Layout für eine Basisdefinition, das durch diese drei Ressourcen dargestellt wird, würde folgendermaßen aussehen:

```
myapp  
└── base  
    ├── deployment.yaml  
    ├── kustomization.yaml  
    ├── secrets.yaml  
    └── service.yaml
```

Durch die Trennung der Kubernetes-Objektdefinition in kleinere Dateien kann der Basissatz aufrecht erhalten werden, wenn er nicht aus einer einzelnen externen Quelle stammt.

Basiskonfiguration

Die Datei `kustomization.yaml` eines Overlays muss auf die Basiskonfigurationssätze verweisen, die den Ausgangspunkt bilden. Beispiel:

```
bases:  
- ../../base
```

Wenn Sie der erwarteten Verzeichnisstruktur folgen, ist der relative Pfad die am besten portierbare Lösung.

Dieses Overlay beginnt mit der Basiskonfiguration und wendet dann alle darin definierten Patches an.

Allgemeine Ressourcen

Mit Kustomize können neue Konfigurationen in allen Ressourcen hinzugefügt werden, die aus einem Basissatz stammen.

Sie können Labels und Annotationen mit den Abschnitten `commonLabels` und `commonAnnotations` einer `kustomization.yaml`-Datei festlegen.

Wenn Sie beispielsweise dem Basissatz das Label `origin=kustomize` und alle davon abhängigen Overlays hinzufügen möchten, müssen Sie dies zu `base/kustomization.yaml` hinzufügen:

```
commonLabels:  
  origin: kustomize
```

Patches

Jedes Overlay kann über die Liste der anzuwendenden Patches eine beliebige Anzahl an Änderungen an der Basiskonfiguration bereitstellen. Fügen Sie zunächst den Verweis auf die Patch-Datei in der Datei `kustomization.yaml` des Overlays hinzu:

```
patches:  
- replica_count.yaml
```

In der bereitgestellten Patch-Datei müssen Sie dann die zu ändernde Ressource und den neuen Wert identifizieren können:

```
apiVersion: apps/v1  
kind: Deployment ①  
metadata:  
  name: myapp ②  
spec:  
  replicas: 5 ③
```

① ② Art und Name bieten eine eindeutige Möglichkeit, die Ressource zu identifizieren

③ Neuer Wert in diesem Overlay

Anwenden von Anpassungen

Sie können Kustomize als ein Standalone-Tool mit dem Befehlszeilentool kustomize oder ab Kubernetes 1.14 als Teil der Befehle kubectl apply oder oc apply verwenden. Diese Tools benötigen nur den Speicherort des Ordners mit dem anzuwendenden Konfigurationssatz.

Hier sehen Sie ein Beispiel für die Standalone-Methode:

```
[user@host ~]$ kustomize build myapp/base | oc apply -f -
```

Dies ist ein Beispiel für die Integration in die Cluster-Management-Tools:

```
[user@host ~]$ oc apply -k myapp/base
```

Und ein Beispiel für die Anwendung der Overlay-Änderungen:

```
[user@host ~]$ oc apply -k myapp/overlays/staging
```

Kustomize ist nur als Teil der Befehle kubectl apply oder oc apply verfügbar, da es sich um eine Erweiterung der deklarativen Verwaltung von Kubernetes-Objekten handelt, die in Kombination mit Versionskontrollsystmen eingesetzt werden soll. Beim deklarativen Verwaltungsstil wird der erwartete Status der Objekte definiert. Ein Objekt kann erstellt oder geändert werden, je nachdem, ob dieses Objekt existiert oder nicht.

Wenn die Quelldatei des Kubernetes-Objekts geändert werden kann, um einen Fehler zu korrigieren oder etwas zu ändern, dann ist es wesentlich ordentlicher und schneller, die Datei zu ändern und oc apply -f myobject.yaml zu verwenden, als sie zu löschen und erneut zu erstellen. Die Änderungen werden auch zur späteren Verwendung dauerhaft in der Datei gespeichert.

Die imperitative Verwaltung von Kubernetes-Objekten verwendet die Befehle kubectl und oc, z. B. create, delete, edit und set, die fehlschlagen, wenn der Vorgang nicht ausgeführt werden kann. Wenn Sie beispielsweise versuchen, ein bereits vorhandenes Objekt zu erstellen, gibt das Tool einen Fehler zurück. Dieser Verwaltungsstil ist für manuelle Vorgänge und Live-Interaktionen mit dem Cluster und nicht für automatisierte Tools vorgesehen.



Literaturhinweise

Kustomize – Kubernetes-natives Konfigurationsmanagement

<https://kustomize.io/>

Kubernetes-Objektverwaltung

<https://kubernetes.io/docs/concepts/overview/working-with-objects/object-management/>

► Angeleitete Übung

Anpassen von Bereitstellungen mit Kustomize

In dieser Übung passen Sie eine Red Hat OpenShift-Bereitstellung mit Kustomize an.

Ergebnisse

Sie sollten in der Lage sein, die Bereitstellung der Anwendung „Famous Quotes“ anzupassen und in einem RHOC-Cluster bereitzustellen.

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen konfigurierten und aktiven RHOC-Cluster

Führen Sie auf dem Rechner `workstation` als Benutzer `student` den Befehl `lab` aus, um die Voraussetzungen für diese Übung zu überprüfen:

```
[student@workstation ~]$ lab multicontainer-kustomize start
```

Anweisungen

Sie richten drei Umgebungen ein, für Entwicklung, Staging und Produktion. Diese drei Umgebungen haben unterschiedliche Anforderungen hinsichtlich Leistung und Anzahl der ausgeführten Pods.

In der folgenden Tabelle finden Sie die Anforderungen der einzelnen Umgebungen:

Umgebung	Pods	Speicher	CPU-Limit
Entwicklung	1	128Mi	250m
Staging	2	256Mi	500m
Produktion	5	512Mi	1000m

► 1. Überprüfen Sie die Deploymentdatei für „Famous Quotes“.

Die Datei `famous-quotes.yaml` enthält alle Ressourcen, die zur Bereitstellung der Anwendung „Famous Quotes“ benötigt werden, die in der angeleiteten Übung zum Erstellen eines Helm-Charts verwendet wird.

```
[student@workstation ~]$ cd ~/D0288/labs/multicontainer-kustomize
[student@workstation multicontainer-kustomize]$ cat famous-quotes.yaml
...
containers:
  - name: famouschart
```

```
securityContext:  
  {}  
image: "quay.io/redhattraining/famous-quotes:2.1"  
...
```

► 2. Erstellen Sie die Ordnerstruktur „Kustomize“.

Um die Ressourcendatei einer Anwendung anzupassen, sollten Sie die Dateien so strukturieren, dass sie leicht zu verstehen und zu referenzieren sind.

- 2.1. Erstellen Sie den Ordner, um alle Kustomize-Dateien darin zu speichern.

```
[student@workstation multicontainer-kustomize]$ mkdir famous-kustomize  
[student@workstation multicontainer-kustomize]$ cd famous-kustomize
```

- 2.2. Erstellen Sie die Basisreferenz.

Verwenden Sie die Datei `famous-quotes.yaml` als Basis für die Kustomize-Struktur.

```
[student@workstation famous-kustomize]$ mkdir base  
[student@workstation famous-kustomize]$ cp ../famous-quotes.yaml \  
base/deployment.yaml
```

- 2.3. Erstellen Sie die Kustomize-Beschreibungsdatei für die Basisdefinition.

Erstellen Sie im Basisordner die neue Datei `kustomization.yaml`, um die Kustomize-Basisdefinition zu speichern. Fügen Sie die Datei `deployment.yaml` als eine Ressource in der Datei `kustomization.yaml` des Ordners `base` hinzu.

```
resources:  
- deployment.yaml
```

► 3. Testen Sie die Basisdefinition.

Zum Anpassen einer Bereitstellung müssen Sie zunächst verifizieren, dass die Basisdefinition erwartungsgemäß funktioniert.

- 3.1. Erstellen Sie ein Projekt in RHOPC.

```
[student@workstation famous-kustomize]$ source /usr/local/etc/ocp4.config  
[student@workstation famous-kustomize]$ oc login -u ${RHT_OCP4_DEV_USER} \  
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}  
[student@workstation famous-kustomize]$ oc new-project \  
${RHT_OCP4_DEV_USER}-multicontainer-kustomize
```

- 3.2. Wenden Sie die Kustomize-Basisdefinition an.

Wenden Sie die Basisdefinition mit dem Befehl `oc apply` an:

```
[student@workstation famous-kustomize]$ oc apply -k base  
serviceaccount/famous-quotes-service created  
serviceaccount/famousapp-mariadb created  
configmap/famousapp-mariadb created  
secret/famousapp-mariadb created
```

```
service/famousapp-famouschart created
service/famousapp-mariadb created
deployment.apps/famousapp-famouschart created
statefulset.apps/famousapp-mariadb created
pod/famousapp-famouschart-test-connection created
```

Mit diesem Befehl wird eine RHOCP-Bereitstellung mit dem Namen **famousapp** erstellt. Führen Sie den Befehl `oc get deployments` aus, um den Status dieser Bereitstellung zu verifizieren:

```
[student@workstation famous-kustomize]$ oc get deployments
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
famousapp-famouschart   1/1     1          1          10s
```

Es kann eine Weile dauern, bis der Pod **mariadb** vollständig für die Anwendung verfügbar ist. Warten Sie daher, bis die Bereitstellung den betriebsbereiten Status erreicht hat.

- 3.3. Stellen Sie den Service der Anwendung bereit:

```
[student@workstation famous-kustomize]$ oc expose service famousapp-famouschart
route.route.openshift.io/famousapp-famouschart exposed
```

- 3.4. Rufen Sie den `/random`-Endpunkt der bereitgestellten Anwendung auf:

```
[student@workstation famous-kustomize]$ FAMOUS_URL=$(oc get route \
-n ${RHT_OCP4_DEV_USER}-multicontainer-kustomize famousapp-famouschart \
-o jsonpath='{.spec.host}'/random)
[student@workstation famous-kustomize]$ curl $FAMOUS_URL
5: Imagination is more important than knowledge.
- Albert Einstein
```

Für jede Anforderung sollte ein zufälliges Zitat angezeigt werden.

- 4. Erstellen Sie die Umgebungsdefinition für die Entwicklungsumgebung.

- 4.1. Erstellen Sie das Entwicklungs-Overlay im Ordner „Kustomize“.

```
[student@workstation famous-kustomize]$ mkdir -p overlays/dev
```

- 4.2. Passen Sie die Richtlinien der Entwicklungsumgebung an.

Erstellen Sie die Datei `replica_limits.yaml` im Verzeichnis `overlays/dev`, um die Änderungen in der Deploymentdefinition zu speichern:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: famousapp-famouschart
spec:
  replicas: 1
  template:
    spec:
      containers:
```

Kapitel 6 | Bereitstellen von Anwendungen mit mehreren Containern

```
- name: famouschart
  resources:
    limits:
      memory: "128Mi"
      cpu: "250m"
```

- 4.3. Erstellen Sie die Datei `overlays/dev/kustomization.yaml`, fügen Sie den Basisordner als Basisdefinition und die Datei `replica_limits.yaml` als Patch hinzu:

```
bases:
- ../../base
patches:
- replica_limits.yaml
```

- 4.4. Wenden Sie das Entwicklungs-Overlay auf die ausgeführte Instanz der Anwendung „Famous Quotes“ an:

```
[student@workstation famous-kustomize]$ oc apply -k overlays/dev
serviceaccount/famous-quotes-service unchanged
serviceaccount/famousapp-mariadb unchanged
configmap/famousapp-mariadb unchanged
secret/famousapp-mariadb unchanged
service/famousapp-famouschart unchanged
service/famousapp-mariadb configured
deployment.apps/famousapp-famouschart configured
statefulset.apps/famousapp-mariadb configured
pod/famousapp-famouschart-test-connection unchanged
```

- 4.5. Überprüfen Sie, ob die Anwendung weiterhin ordnungsgemäß funktioniert:

```
[student@workstation famous-kustomize]$ curl $FAMOUS_URL
4: Nothing that glitters is gold.
- Mark Twain
```

- 4.6. Überprüfen Sie, ob Kustomize die Änderung der Arbeitsspeicherbeschränkung angewendet hat:

```
[student@workstation famous-kustomize]$ oc get deployments famousapp-famouschart \
-o jsonpath='{.spec.template.spec.containers[0].resources.limits.memory}'
128Mi
```

- 5. Erstellen Sie die Staging-Umgebung.

- 5.1. Erstellen Sie die Staging-Umgebung, indem Sie die Entwicklungsumgebung kopieren:

```
[student@workstation famous-kustomize]$ cp -R overlays/dev overlays/stage
```

- 5.2. Ändern Sie die Konfiguration in der Datei `overlays/stage/replica_limits.yaml` entsprechend den Werten in der Tabelle für Anforderungen:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: famousapp-famouschart
spec:
  replicas: 2
  template:
    spec:
      containers:
        - name: famouschart
          resources:
            limits:
              memory: "256Mi"
              cpu: "500m"
```

- 5.3. Wenden Sie das Staging-Overlay auf die ausgeführte Instanz der Anwendung „Famous Quotes“ an:

```
[student@workstation famous-kustomize]$ oc apply -k overlays/stage
serviceaccount/famous-quotes-service unchanged
serviceaccount/famousapp-mariadb unchanged
configmap/famousapp-mariadb unchanged
secret/famousapp-mariadb unchanged
service/famousapp-famouschart unchanged
service/famousapp-mariadb configured
deployment.apps/famousapp-famouschart configured
statefulset.apps/famousapp-mariadb configured
pod/famousapp-famouschart-test-connection unchanged
```

- 5.4. Überprüfen Sie, ob die Anwendung weiterhin ordnungsgemäß funktioniert:

```
[student@workstation famous-kustomize]$ curl $FAMOUS_URL
2: Happiness depends upon ourselves.
- Aristotle
```

- 5.5. Überprüfen Sie, ob Kustomize die Änderung der Arbeitsspeicherbeschränkung angewendet hat:

```
[student@workstation famous-kustomize]$ oc get deployments famousapp-famouschart \
-o jsonpath='{.spec.template.spec.containers[0].resources.limits.memory}'
256Mi
```

- 6. Erstellen Sie die Produktionsumgebung.

- 6.1. Erstellen Sie die Produktionsumgebung, indem Sie die Entwicklungsumgebung kopieren:

```
[student@workstation famous-kustomize]$ cp -R overlays/dev overlays/prod
```

Kapitel 6 | Bereitstellen von Anwendungen mit mehreren Containern

- 6.2. Ändern Sie die Konfiguration in der Datei `overlays/prod/relica_limits.yaml` entsprechend den Werten in der Tabelle für Anforderungen:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: famousapp-famouschart
spec:
  replicas: 5
  template:
    spec:
      containers:
        - name: famouschart
          resources:
            limits:
              memory: "512Mi"
              cpu: "1000m"
```

- 6.3. Wenden Sie das Produktions-Overlay auf die ausgeführte Instanz der Anwendung „Famous Quotes“ an:

```
[student@workstation famous-kustomize]$ oc apply -k overlays/prod
serviceaccount/famous-quotes-service unchanged
serviceaccount/famousapp-mariadb unchanged
configmap/famousapp-mariadb unchanged
secret/famousapp-mariadb unchanged
service/famousapp-famouschart unchanged
service/famousapp-mariadb configured
deployment.apps/famousapp-famouschart configured
statefulset.apps/famousapp-mariadb configured
pod/famousapp-famouschart-test-connection unchanged
```

- 6.4. Überprüfen Sie, ob die Anwendung weiterhin ordnungsgemäß funktioniert:

```
[student@workstation famous-kustomize]$ curl $FAMOUS_URL
8: Those who can imagine anything, can create the impossible.
- Alan Turing
```

- 6.5. Überprüfen Sie, ob Kustomize die Änderung der Arbeitsspeicherbeschränkung angewendet hat:

```
[student@workstation famous-kustomize]$ oc get deployments famousapp-famouschart \
-o jsonpath='{.spec.template.spec.containers[0].resources.limits.memory}'
512Mi
```

Beenden

Führen Sie auf der `workstation`-VM den Befehl `lab multicontainer-kustomize finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken. Die Aktion „Beenden“ veröffentlicht dieses Projekt und seine Ressourcen.

```
[student@workstation famous-kustomize]$ cd ~  
[student@workstation ~]$ lab multicontainer-kustomize finish
```

Hiermit ist die angeleitete Übung beendet.

► Praktische Übung

Bereitstellen von Anwendungen mit mehreren Containern

In dieser Übung erstellen Sie ein Helm-Chart für eine Anwendung und stellen es in einem OpenShift-Cluster bereit. Anschließend passen Sie die bereitgestellte Anwendung mithilfe von Kustomize an.



Anmerkung

Für den Befehl grade am Ende jeder praktischen Übung eines Kapitels ist es erforderlich, dass Sie die exakten Projektnamen und anderen Identifikatoren, wie in der Spezifikation der praktischen Übung angegeben, verwenden.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Erstellen eines Helm-Charts für eine Anwendung
- Bereitstellen der Anwendung auf einem OpenShift-Cluster
- Verwenden von Kustomize, um die Bereitstellung der Anwendung zu ändern

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen aktiven OpenShift-Cluster
- Auf ein Anwendungscontainer-Image in quay.io/redhattraining/exoplanets:v1.0
- Auf ein CockroachDB-Helm-Chart aus dem Repository <https://charts.cockroachdb.com/>

Führen Sie den folgenden Befehl auf workstation aus, um die Voraussetzungen zu überprüfen. Der Befehl lädt auch die Hilfs- und Lösungsdateien für die Wiederholungsübung herunter:

```
[student@workstation ~]$ lab multicontainer-review start
```

Anforderungen

Die Anwendung „Exoplanets“ ist eine Webanwendung, die Informationen über extrasolare Planeten zeigt. Diese Anwendung benötigt eine mit PostgreSQL kompatible Datenbank, beispielsweise CockroachDB, um die Informationen zu speichern, und sie verfügt über ein vorgefertigtes Container-Image in Quay. Verwenden Sie für den Zugriff auf die Webanwendung einen Webbrowser, der auf den bereitgestellten Service verweist.

Erstellen Sie das Helm-Chart für die Exoplanets-Anwendung, und stellen Sie es entsprechend den folgenden Anforderungen auf dem OpenShift-Cluster bereit:

- Verwenden Sie `exochart` als Namen für das Helm-Chart.
- Verwenden Sie den Anwendungscontainer in `quay.io/redhattraining/exoplanets:v1.0`.
- Verwenden Sie das Helm-Chart `cockroachdb` mit der Version `6.0.4` aus dem Repository <https://charts.cockroachdb.com/> als Datenbankabhängigkeit.
- Verwenden Sie Port `8080` für die Anwendung.
- Verwenden Sie die folgenden Umgebungsvariablen, um die Datenbankverbindung zu konfigurieren:

Umgebungsvariablen für die Datenbankkonfiguration

Variable	Wert
DB_HOST	exoplanets-cockroachdb
DB_PORT	26257
DB_USER	root
DB_NAME	postgres

- Verwenden Sie `youruser-multicontainer-review` als RHOCP-Projektnamen.
- Verwenden Sie `exoplanets` als Installationsnamen des Helm-Charts.
- Verwenden Sie `exokustom` als Kustomize-Verzeichnis.
- Verwenden Sie den Befehl `helm template`, um die Kustomize-Basisdefinition aus dem Helm-Chart zu erstellen.
- Verwenden Sie `test` als Verzeichnis für das Test-Overlay.
- Verwenden Sie die folgenden Ressourcenlimits für das Test-Overlay:

Umgebung	Pods	Speicher	CPU-Limit
Test	5	128Mi	250m

Anweisungen

1. Erstellen Sie ein Helm-Chart für die Anwendung „Exoplanets“ mit dem Namen `exochart`.
2. Fügen Sie die Datenbankabhängigkeit hinzu, und konfigurieren Sie die Datenbank mithilfe von Umgebungsvariablen.
3. Erstellen Sie ein OpenShift-Projekt, und stellen Sie die Anwendung mit dem Helm-CLI-Tool im Projekt bereit.
4. Testen Sie die Anwendungsbereitstellung.

Stellen Sie den Anwendungsservice bereit, rufen Sie die Adresse aus der Route der Anwendung ab, und navigieren Sie mit einem Browser zur Adresse der Anwendung. Eine ordnungsgemäße Bereitstellung der Anwendung ähnelt diesem Image:

Exoplanets

The planets listed here are a small subset of the known planets found outside of our solar system. Mass and radius are listed in "Jupiter mass" and "Jupiter radius" units. The orbital period is measured in Earth days. The full dataset is available from the [Open Exoplanet Catalogue](#).

2M 0746+20 b

Mass	Radius	Period
30	0.97	4640

2M 2140+16 b

Mass	Radius	Period
20	0.92	7340

- Erstellen Sie die Kustomize-Basisverzeichnisstruktur und die Deploymentdateien. Erstellen Sie das Kustomize-Verzeichnis mit dem Namen `exokustom` im Verzeichnis `~/D0288/labs/multicontainer-review`.



Anmerkung

Sie können den Befehl `helm template` verwenden, um die Objektdefinitionen aus einem-Helm Chart in die Basisdefinition zu extrahieren:

```
[student@workstation ~]$ helm template app-name helm-directory >
base/deployment.yaml
```

- Erstellen Sie das Test-Overlay im Kustomize-Verzeichnis unter Verwendung des Verzeichnisses `test`, und wenden Sie es an.
- Testen Sie die Anwendung im Browser, und überprüfen Sie, ob die benutzerdefinierten Werte angewendet wurden.

Bewertung

Verwenden Sie als Benutzer `student` auf dem Rechner `workstation` den Befehl `lab`, um Ihre Arbeit zu bewerten. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie den Befehl so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation exokustom]$ lab multicontainer-review grade
```

Beenden

Führen Sie auf `workstation` den Befehl `lab multicontainer-review finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken.

```
[student@workstation exokustom]$ cd  
[student@workstation ~]$ lab multicontainer-review finish
```

Hiermit ist die praktische Übung abgeschlossen.

► Lösung

Bereitstellen von Anwendungen mit mehreren Containern

In dieser Übung erstellen Sie ein Helm-Chart für eine Anwendung und stellen es in einem OpenShift-Cluster bereit. Anschließend passen Sie die bereitgestellte Anwendung mithilfe von Kustomize an.



Anmerkung

Für den Befehl grade am Ende jeder praktischen Übung eines Kapitels ist es erforderlich, dass Sie die exakten Projektnamen und anderen Identifikatoren, wie in der Spezifikation der praktischen Übung angegeben, verwenden.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Erstellen eines Helm-Charts für eine Anwendung
- Bereitstellen der Anwendung auf einem OpenShift-Cluster
- Verwenden von Kustomize, um die Bereitstellung der Anwendung zu ändern

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen aktiven OpenShift-Cluster
- Auf ein Anwendungscontainer-Image in quay.io/redhattraining/exoplanets:v1.0
- Auf ein CockroachDB-Helm-Chart aus dem Repository <https://charts.cockroachdb.com/>

Führen Sie den folgenden Befehl auf workstation aus, um die Voraussetzungen zu überprüfen. Der Befehl lädt auch die Hilfs- und Lösungsdateien für die Wiederholungsübung herunter:

```
[student@workstation ~]$ lab multicontainer-review start
```

Anforderungen

Die Anwendung „Exoplanets“ ist eine Webanwendung, die Informationen über extrasolare Planeten zeigt. Diese Anwendung benötigt eine mit PostgreSQL kompatible Datenbank, beispielsweise CockroachDB, um die Informationen zu speichern, und sie verfügt über ein vorgefertigtes Container-Image in Quay. Verwenden Sie für den Zugriff auf die Webanwendung einen Webbrowser, der auf den bereitgestellten Service verweist.

Erstellen Sie das Helm-Chart für die Exoplanets-Anwendung, und stellen Sie es entsprechend den folgenden Anforderungen auf dem OpenShift-Cluster bereit:

- Verwenden Sie `exochart` als Namen für das Helm-Chart.
- Verwenden Sie den Anwendungscontainer in `quay.io/redhattraining/exoplanets:v1.0`.
- Verwenden Sie das Helm-Chart `cockroachdb` mit der Version `6.0.4` aus dem Repository `https://charts.cockroachdb.com/` als Datenbankabhängigkeit.
- Verwenden Sie Port `8080` für die Anwendung.
- Verwenden Sie die folgenden Umgebungsvariablen, um die Datenbankverbindung zu konfigurieren:

Umgebungsvariablen für die Datenbankkonfiguration

Variable	Wert
DB_HOST	exoplanets-cockroachdb
DB_PORT	26257
DB_USER	root
DB_NAME	postgres

- Verwenden Sie `youruser-multicontainer-review` als RHOCP-Projektnamen.
- Verwenden Sie `exoplanets` als Installationsnamen des Helm-Charts.
- Verwenden Sie `exokustom` als Kustomize-Verzeichnis.
- Verwenden Sie den Befehl `helm template`, um die Kustomize-Basisdefinition aus dem Helm-Chart zu erstellen.
- Verwenden Sie `test` als Verzeichnis für das Test-Overlay.
- Verwenden Sie die folgenden Ressourcenlimits für das Test-Overlay:

Umgebung	Pods	Speicher	CPU-Limit
Test	5	128Mi	250m

Anweisungen

1. Erstellen Sie ein Helm-Chart für die Anwendung „Exoplanets“ mit dem Namen `exochart`.
 - 1.1. Erstellen Sie das-Helm Chart `exochart`.
Erstellen Sie das Helm-Chart, indem Sie den Befehl `helm create` verwenden, und nennen Sie es `exochart`.

```
[student@workstation ~]$ cd ~/DO288/labs/multicontainer-review
[student@workstation multicontainer-review]$ helm create exochart
Creating exochart
[student@workstation multicontainer-review]$ cd exochart
```

Kapitel 6 | Bereitstellen von Anwendungen mit mehreren Containern

- 1.2. Konfigurieren Sie Image- und Versionswerte für den Anwendungscontainer.

Aktualisieren Sie die Datei `values.yaml`. Legen Sie dabei die Eigenschaft `repository` des Abschnitts `image` auf das Image `quay.io/redhattraining/exoplanets` und die `tag`-Eigenschaft desselben Abschnitts auf den Wert `v1.0` fest, um die entsprechende Version der Anwendung auszuwählen.

Der entsprechende Abschnitt der Datei `values.yaml` sollte folgendermaßen aussehen:

```
image:
  repository: quay.io/redhattraining/exoplanets
  pullPolicy: IfNotPresent
  tag: "v1.0"
```

2. Fügen Sie die Datenbankabhängigkeit hinzu, und konfigurieren Sie die Datenbank mithilfe von Umgebungsvariablen.

- 2.1. Fügen Sie dem Anwendungsdiagramm die Abhängigkeit `cockroachdb` hinzu.

Fügen Sie dazu den folgenden Ausschnitt am Ende der Datei `Chart.yaml` hinzu:

```
dependencies:
- name: cockroachdb
  version: 6.0.4
  repository: https://charts.cockroachdb.com/
```

- 2.2. Aktualisieren Sie die Abhängigkeiten für das Diagramm.

Mit diesem Befehl werden die als Abhängigkeiten hinzugefügten Diagramme heruntergeladen und deren Versionen gesperrt.

```
[student@workstation exochart]$ helm dependency update
Getting updates for unmanaged Helm repositories...
...Successfully got an update from the "https://charts.cockroachdb.com/" chart
repository
Saving 1 charts
Downloading cockroachdb from repo https://charts.cockroachdb.com/
Deleting outdated charts
```

- 2.3. Die StandardDeploymentvorlage übergibt keine Umgebungsvariablen an die bereitgestellten Anwendungen. Ändern Sie die Vorlage `templates/deployment.yaml`, um die in der Datei `values.yaml` definierten Umgebungsvariablen an den Container der Anwendung zu übergeben. Fügen Sie nach dem Wert `imagePullPolicy` im Abschnitt `containers` den folgenden Ausschnitt hinzu:

```
apiVersion: apps/v1
kind: Deployment
...output omitted...
spec:
  ...output omitted...
  template:
    ...output omitted...
    spec:
      ...output omitted...
```

```

containers:
  - name: {{ .Chart.Name }}
    ...output omitted...
    imagePullPolicy: {{ .Values.image.pullPolicy }}
    env:
      {{- range .Values.env }}
      - name: "{{ .name }}"
        value: "{{ .value }}"
      {{- end }}
    ...output omitted...
  
```

- 2.4. Stellen Sie sicher, dass der Container den richtigen Port verwendet, um eine Verbindung zur Anwendung herzustellen.

Ändern Sie in der Datei `templates/deployment.yaml` die Eigenschaft `containerPort` im Abschnitt `containers` so, dass der Wert `8080` verwendet wird.

Der entsprechende Abschnitt der Datei `templates/deployment.yaml` sollte folgendermaßen aussehen:

```

apiVersion: apps/v1
kind: Deployment
...output omitted...
spec:
  ...output omitted...
  template:
    ...output omitted...
    spec:
      ...output omitted...
      containers:
        - name: {{ .Chart.Name }}
          ...output omitted...
          ports:
            - name: http
              containerPort: 8080
              protocol: TCP
  
```

- 2.5. Fügen Sie am Ende der Datei `values.yaml` die entsprechenden Umgebungsvariablen hinzu:

```

env:
  - name: "DB_HOST"
    value: "exoplanets-cockroachdb"
  - name: "DB_NAME"
    value: "postgres"
  - name: "DB_USER"
    value: "root"
  - name: "DB_PORT"
    value: "26257"
  
```

3. Erstellen Sie ein OpenShift-Projekt, und stellen Sie die Anwendung mit dem Helm-CLI-Tool im Projekt bereit.

- 3.1. Erstellen Sie ein Projekt in OpenShift.

Kapitel 6 | Bereitstellen von Anwendungen mit mehreren Containern

```
[student@workstation exochart]$ source /usr/local/etc/ocp4.config
[student@workstation exochart]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
[student@workstation exochart]$ oc new-project \
${RHT_OCP4_DEV_USER}-multicontainer-review
```

- 3.2. Führen Sie den Befehl `helm install` aus, um die Anwendung im OpenShift-Cluster bereitzustellen:

```
[student@workstation exochart]$ helm install exoplanets .
NAME: exoplanets
LAST DEPLOYED: Thu May 20 19:12:09 2021
NAMESPACE: youruser-multicontainer-review
STATUS: deployed
REVISION: 1
NOTES:
...output omitted...
```

Überprüfen Sie, ob die Bereitstellung der Anwendung ordnungsgemäß gestartet wurde:

```
[student@workstation exochart]$ oc get deployments
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
exoplanets-exochart   0/1       1           1          10s
```

Warten Sie, bis die drei cockroachdb- und Anwendungs-Pods vollständig für die Anwendung verfügbar sind. Verwenden Sie mehrmals den Befehl `oc get pods`, um zu verifizieren, dass die Anwendung und die Datenbank ordnungsgemäß bereitgestellt wurden:

```
[student@workstation exochart]$ oc get pods
NAME                           READY   STATUS    RESTARTS   AGE
exoplanets-cockroachdb-0      1/1     Running   0          1m
exoplanets-cockroachdb-1      1/1     Running   0          1m
exoplanets-cockroachdb-2      1/1     Running   0          1m
exoplanets-exochart-7bff544d88-f289l   1/1     Running   3          1m
```

4. Testen Sie die Anwendungsbereitstellung.

Stellen Sie den Anwendungsservice bereit, rufen Sie die Adresse aus der Route der Anwendung ab, und navigieren Sie mit einem Browser zur Adresse der Anwendung. Eine ordnungsgemäße Bereitstellung der Anwendung ähnelt diesem Image:

Exoplanets

The planets listed here are a small subset of the known planets found outside of our solar system. Mass and radius are listed in "Jupiter mass" and "Jupiter radius" units. The orbital period is measured in Earth days. The full dataset is available from the [Open Exoplanet Catalogue](#).

2M 0746+20 b

Mass	Radius	Period
30	0.97	4640

2M 2140+16 b

Mass	Radius	Period
20	0.92	7340

- 4.1. Stellen Sie den Service der Anwendung bereit:

```
[student@workstation exochart]$ oc expose service exoplanets-exochart  
route.route.openshift.io/exoplanets-exochart exposed
```

- 4.2. Öffnen Sie die bereitgestellte Anwendung in einem Browser:

```
[student@workstation exochart]$ firefox $(oc get route exoplanets-exochart \  
-o jsonpath='{.spec.host}' \  
-n ${RHT_OCP4_DEV_USER}-multicontainer-review ) &
```

5. Erstellen Sie die Kustomize-Basisverzeichnisstruktur und die Deploymentdateien.

Erstellen Sie das Kustomize-Verzeichnis mit dem Namen `exokustom` im Verzeichnis `~/D0288/labs/multicontainer-review`.



Anmerkung

Sie können den Befehl `helm template` verwenden, um die Objektdefinitionen aus einem-Helm Chart in die Basisdefinition zu extrahieren:

```
[student@workstation ~]$ helm template app-name helm-directory >  
base/deployment.yaml
```

- 5.1. Erstellen Sie das Verzeichnis `exokustom`, in dem alle Kustomize-Verzeichnisse gespeichert werden.

```
[student@workstation exochart]$ cd ..  
[student@workstation multicontainer-review]$ mkdir exokustom  
[student@workstation exochart]$ cd exokustom
```

- 5.2. Erstellen Sie das Verzeichnis für die Basisdefinition.

```
[student@workstation exokustom]$ mkdir base
```

- 5.3. Verwenden Sie den Befehl `helm template`, um die Objektdefinitionen aus einem Helm-Chart in die Basisdefinition zu extrahieren:

```
[student@workstation exokustom]$ helm template exoplanets \  
..../exochart > base/deployment.yaml
```

- 5.4. Erstellen Sie die Kustomize-Beschreibungsdatei für die Basisdefinition.

Erstellen Sie im Verzeichnis `base` die neue Datei `kustomization.yaml`, um die Kustomize-Basisdefinition zu speichern. Fügen Sie die Datei `deployment.yaml` als eine Ressource in der Datei `kustomization.yaml` des Verzeichnisses `Base` hinzu.

```
resources:  
- deployment.yaml
```

6. Erstellen Sie das Test-Overlay im Kustomize-Verzeichnis unter Verwendung des Verzeichnisses `test`, und wenden Sie es an.

- 6.1. Erstellen Sie das Test-Overlay im Kustomize-Verzeichnis.

```
[student@workstation exokustom]$ mkdir -p overlays/test
```

- 6.2. Passen Sie die Richtlinien der Testumgebung an.

Erstellen Sie die Datei `replica_limits.yaml` im Verzeichnis `overlays/test`, um die Änderungen in der Deploymentdefinition zu speichern: Diese Änderungen müssen die in den Anforderungen der praktischen Übung definierten Ressourcenlimits widerspiegeln:

```
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: exoplanets-exochart  
spec:  
  replicas: 5  
  template:  
    spec:  
      containers:  
        - name: exochart  
          resources:  
            limits:  
              memory: "128Mi"  
              cpu: "250m"
```

- 6.3. Erstellen Sie die Datei `overlays/test/kustomization.yaml`, fügen Sie das Basisverzeichnis als Basisdefinition und die Datei `replica_limits.yaml` als Patch hinzu:

```
bases:  
- ../../base  
patches:  
- replica_limits.yaml
```

- 6.4. Wenden Sie das Test-Overlay auf die ausgeführte Instanz der Anwendung „Exoplanets“ an:

```
[student@workstation exokustom]$ oc apply -k overlays/test  
...output omitted...  
serviceaccount/exoplanets-exochart configured  
service/exoplanets-cockroachdb-public configured  
service/exoplanets-cockroachdb configured  
service/exoplanets-exochart configured  
deployment.apps/exoplanets-exochart configured  
statefulset.apps/exoplanets-cockroachdb configured  
...output omitted...
```

Warnungen dieser Form können ignoriert werden:

```
Warning: oc apply should be used on resource created by either oc create --save-config or oc apply
```

7. Testen Sie die Anwendung im Browser, und überprüfen Sie, ob die benutzerdefinierten Werte angewendet wurden.
- 7.1. Überprüfen Sie, ob die Anwendung weiterhin ordnungsgemäß funktioniert.
Aktualisieren Sie den Browser. Dieselbe Anwendung sollte weiterhin angezeigt werden.
 - 7.2. Überprüfen Sie, ob Kustomize die Änderung von CPU und Arbeitsspeicherbeschränkung angewendet hat:

```
[student@workstation exokustom]$ oc get deployments exoplanets-exochart \  
-o jsonpath='{.spec.template.spec.containers[0].resources.limits}'  
{"cpu":"250m", "memory":"128Mi"}
```

Bewertung

Verwenden Sie als Benutzer `student` auf dem Rechner `workstation` den Befehl `lab`, um Ihre Arbeit zu bewerten. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie den Befehl so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation exokustom]$ lab multicontainer-review grade
```

Beenden

Führen Sie auf `workstation` den Befehl `lab multicontainer-review finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken.

```
[student@workstation exokustom]$ cd  
[student@workstation ~]$ lab multicontainer-review finish
```

Hiermit ist die praktische Übung abgeschlossen.

Zusammenfassung

In diesem Kapitel wurden die folgenden Themen behandelt:

- Konvertieren einer Reihe von Red Hat OpenShift-Ressourcen in Vorlagen
- Verwenden von Helm-Charts zum Packen von Kubernetes-Anwendungen.
- Anpassen von Kubernetes-Ressourcen für unterschiedliche Umgebungen mit Kustomize
- Verwenden von Kustomize zum Anpassen einer paketierten Kubernetes-Anwendung, ohne sie zu ändern

Kapitel 7

Verwalten der Anwendungsbereitstellungen

Ziel

Überwachen des Anwendungszustands und Implementieren verschiedener Deploymentmethoden für Cloud-native Anwendungen

Ziele

- Implementieren von Liveness-Probes und Readiness-Probes
- Auswählen der geeigneten Deploymentstrategie für Cloud-native Anwendungen
- Verwalten der Bereitstellung einer Anwendung mithilfe von CLI-Befehlen

Abschnitte

- Überwachen des Anwendungszustands (und angeleitete Übung)
- Auswählen einer geeigneten Deploymentstrategie (und angeleitete Übung)
- Verwalten von Anwendungsbereitstellungen mithilfe von CLI-Befehlen (und angeleitete Übung)

Praktische Übung

Verwalten der Anwendungsbereitstellungen

Überwachen des Anwendungszustands

Ziele

In diesem Abschnitt wird beschrieben, wie Startup-, Readiness- und Liveness-Probes zum Überwachen der Bereitschaft und des Zustands von Anwendungen implementiert werden.

Readiness- und Liveness-Probes für Red Hat OpenShift Container Platform

Anwendungen können aus verschiedenen Gründen unzuverlässig werden, beispielsweise:

- Temporärer Verbindungsverlust
- Konfigurationsfehler
- Anwendungsfehler

Entwickler können mithilfe von *Tests* ihre Anwendungen überwachen. Durch Tests werden Entwickler auf Ereignisse wie Anwendungsstatus, Ressourcennutzung und Fehler aufmerksam gemacht.

Die Überwachung solcher Ereignisse ist nützlich, um Probleme zu beheben, sie kann aber auch die Ressourcenplanung und -verwaltung unterstützen.

Bei einem Test handelt es sich um eine periodische Überprüfung, bei der der Zustand einer Anwendung überwacht wird. Entwickler können Tests mithilfe des oc-Befehlszeilen-Clients, einer YAML-Deploymentvorlage oder über die Red Hat OpenShift-Web-Konsole konfigurieren.

Derzeit gibt es drei Testtypen in Red Hat OpenShift:

Startup-Test

Ein Startup-Test überprüft, ob die Anwendung in einem Container gestartet wurde. Ein Startup-Test wird vor allen anderen Tests ausgeführt und deaktiviert andere Tests, sofern er nicht erfolgreich abgeschlossen wird. Wenn der Startup-Test für einen Container nicht bestanden wird, wird der Container beendet und folgt der `restartPolicy` des Pods.

Dieser Testtyp wird im Gegensatz zu Readiness-Tests, die regelmäßig ausgeführt werden, nur beim Start ausgeführt.

Der Startup-Test wird im Attribut „spec.containers.startupprobe“ der Pod-Konfiguration konfiguriert.

Readiness-Test

Mithilfe von Readiness-Tests wird ermittelt, ob ein Container Anforderungen verarbeiten kann. Wenn der Readiness-Test einen fehlerhaften Status ermittelt, entfernt Red Hat OpenShift die IP-Adresse des Containers von den Endpunkten aller Services.

Entwickler können anhand von Readiness-Tests Red Hat OpenShift signalisieren, dass ein Container – obwohl er ausgeführt wird – keinen Datenverkehr von einem Proxy empfangen sollte. Dies kann nützlich sein, um darauf zu warten, dass eine Anwendung Netzwerkverbindungen ausführt, Dateien und Cache lädt, oder im Allgemeinen auf

alle anfänglichen Aufgaben, die beträchtliche Zeit in Anspruch nehmen und sich nur vorübergehend auf die Anwendung auswirken können.

Der Readiness-Test wird im Attribut `spec.containers.readinessprobe` der Pod-Konfiguration konfiguriert.

Liveness-Test

Mithilfe von Liveness-Probes wird ermittelt, ob eine Anwendung, die in einem Container ausgeführt ist, sich in einem fehlerfreien Zustand (`healthy`) befindet. Falls der Liveness-Probe einen fehlerhaften Status ermittelt, beendet Red Hat OpenShift den Container und versucht, ihn erneut bereitzustellen.

Der Liveness-Probe wird im Attribut `spec.containers.livenessprobe` der Pod-Konfiguration konfiguriert.

Red Hat OpenShift stellt fünf Optionen zum Steuern dieser Probes zur Verfügung:

Name	Obligatorisch	Beschreibung	Standardwert
<code>initialDelaySeconds</code>	Ja	Legt die Wartezeit nach dem Starten des Containers fest, bevor der Probe initiiert wird.	0
<code>timeoutSeconds</code>	Ja	Legt die Wartezeit bis zum Abschluss des Probes fest. Wenn diese Zeit überschritten wird, nimmt Red Hat OpenShift an, dass der Probe fehlgeschlagen ist.	1
<code>periodSeconds</code>	Nein	Legt die Häufigkeit der Überprüfungen fest.	1
<code>successThreshold</code>	Nein	Legt die Mindestanzahl von aufeinanderfolgenden erfolgreichen Versuchen fest, nach denen der Probe nach einem Fehlschlag als erfolgreich eingestuft werden kann.	1
<code>failureThreshold</code>	Nein	Legt die Mindestanzahl von aufeinanderfolgenden fehlgeschlagenen Versuchen fest, nach denen der Probe nach einer erfolgreichen Ausführung als fehlgeschlagen eingestuft werden kann.	3

Methoden zur Überprüfung des Anwendungszustands

Mithilfe von Startup-, Readiness- und Liveness-Probes kann der Zustand von Anwendungen auf dreierlei Weise überprüft werden:

HTTP-Überprüfungen

HTTP-Überprüfungen eignen sich insbesondere für Anwendungen, die HTTP-Statuscodes zurückgeben, beispielsweise REST-APIs.

Beim HTTP-Probe werden GET-Anforderungen verwendet, um den Status einer Anwendung zu überprüfen. Die Überprüfung ist erfolgreich, wenn der HTTP-Antwortcode im Bereich zwischen 200 und 399 liegt.

Im folgenden Beispiel wird demonstriert, wie ein Readiness-Probe in Kombination mit der HTTP-Prüfmethode implementiert wird:

```
...contents omitted...
readinessProbe:
  httpGet:
    path: /health❶
    port: 8080
    initialDelaySeconds: 15❷
    timeoutSeconds: 1❸
  ...contents omitted...
```

- ❶ Der Readiness-Probe-Endpunkt
- ❷ Die Wartezeit nach dem Starten des Containers, bevor dessen Zustand überprüft wird.
- ❸ Die Wartezeit bis zum Abschluss des Probes.

Überprüfungen der Containerausführung

Überprüfungen der Containerausführung eignen sich ideal in Szenarien, in denen Sie den Status des Containers anhand des Beendigungscodes eines Prozesses oder Shell-Skripts bestimmen müssen, das im Container ausgeführt wird.

Beim Verwenden von Überprüfungen der Containerausführung führt Red Hat OpenShift einen Befehl im Container aus. Wenn die Überprüfung mit dem Status 0 abgeschlossen wird, wird sie als erfolgreich eingestuft. Alle anderen Statuscodes werden als ein Fehler erachtet. Im folgenden Beispiel wird demonstriert, wie die Überprüfung der Containerausführung implementiert wird:

```
...contents omitted...
livenessProbe:
  exec:
    command:❶
      - cat
      - /tmp/health
    initialDelaySeconds: 15
    timeoutSeconds: 1
  ...contents omitted...
```

- ❶ Der auszuführende Befehl und seine Argumente als ein YAML-Array

TCP-Socket-Überprüfungen

Eine TCP-Socket-Überprüfung eignet sich ideal für als Daemons ausgeführte Anwendungen und offene TCP-Ports, beispielsweise Datenbankserver, Dateiserver, Webserver und Anwendungsserver.

Kapitel 7 | Verwalten der Anwendungsbereitstellungen

Beim Verwenden von TCP-Socket-Überprüfungen versucht Red Hat OpenShift, einen Socket im Container zu öffnen. Der Container wird als fehlerfrei betrachtet, wenn bei der Überprüfung eine erfolgreiche Verbindung hergestellt werden kann. Im folgenden Beispiel wird demonstriert, wie ein Liveness-Probe mithilfe der Methode zur TCP-Socket-Überprüfung implementiert wird:

```
...contents omitted...
livenessProbe:
  tcpSocket:
    port: 8080①
    initialDelaySeconds: 15
    timeoutSeconds: 1
...contents omitted...
```

- ① Der zur überprüfende TCP-Port.

Verwalten von Probes über die Web-Konsole

Entwickler können Probes erstellen, indem Sie die YAML-Deploymentdatei mit `oc edit` oder über die Red Hat OpenShift-Web-Konsole bearbeiten.

Sie können die YAML-Deploymentdatei direkt auf der Seite **Workloads → Deployment → <deployment name>** in der Web-Konsole bearbeiten. Klicken Sie auf das Dropdown-Menü **Actions**, und wählen Sie **Edit Deployment** aus.

The screenshot shows the Red Hat OpenShift Web Console interface. At the top, there's a breadcrumb navigation: Deployments > Deployment Details. Below it, a title bar says 'D example'. The main area is titled 'Deployment Details' and shows a summary: 1 pod. On the left, there are tabs for Details, YAML, Replica Sets, Pods, Environment, and Events. The 'Details' tab is selected. On the right, there's a 'Actions' dropdown menu with several options: Edit Pod Count, Pause Rollouts, Add Health Checks, Add Horizontal Pod Autoscaler, Add Storage, Edit Update Strategy, Edit Labels, Edit Annotations, Edit Deployment (which is highlighted with a red box), and Delete Deployment.

Abbildung 7.1: Hinzufügen von Probes mit der Web-Konsole

Im folgenden Beispiel wird der YAML-Editor in der Web-Konsole für eine Bereitstellung dargestellt.

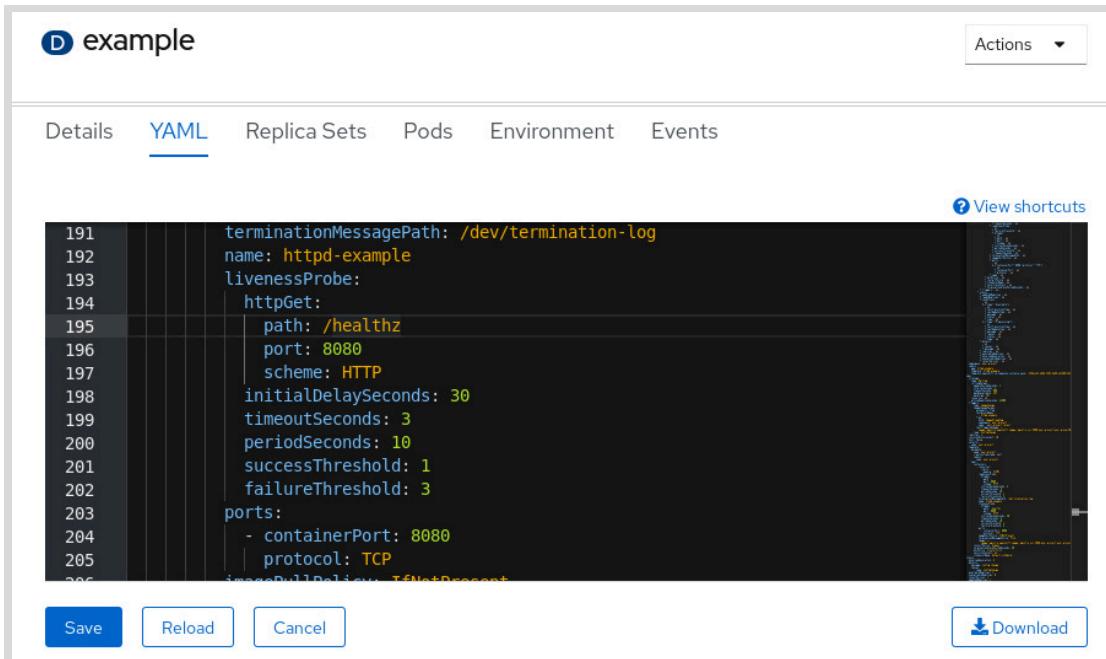


Abbildung 7.2: Hinzufügen von Probes über den YAML-Editor der Web-Konsole

Erstellen von Probes mithilfe der CLI

Der Befehl `oc set probe` bietet einen alternativen Ansatz zum direkten Bearbeiten der YAML-Definition der Bereitstellung. Wenn Sie Probes für die Ausführung von Anwendungen erstellen, verwenden Sie den Befehl `oc set probe`, da er Ihre Fehlerfähigkeit einschränkt. Dieser Befehl bietet eine Reihe von Optionen, mit denen Sie den Probe-Typ sowie andere erforderliche Attribute wie Port, URL, Timeout, Zeitraum und mehr angeben können.

Die folgenden Beispiele veranschaulichen die Verwendung des Befehls `oc set probe` mit einer Vielzahl von Optionen:

```
[user@host ~]$ oc set probe deployment myapp --readiness \
--get-url=http://:8080/healthz --period=20
```

```
[user@host ~]$ oc set probe deployment myapp --liveness \
--open-tcp=3306 --period=20 \
--timeout-seconds=1
```

```
[user@host ~]$ oc set probe deployment myapp --liveness \
--get-url=http://:8080/healthz --initial-delay-seconds=30 \
--success-threshold=1 --failure-threshold=3
```

Führen Sie den Befehl `oc set probe --help` aus, um alle verfügbaren Optionen für diesen Befehl anzuzeigen.



Literaturhinweise

Weitere Informationen finden Sie im Abschnitt *Monitoring application health by using health checks* im Kapitel *Applications* der offiziellen Dokumentation für Red Hat OpenShift Container Platform 4.6 unter

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/applications/index#application-health

Weitere Informationen finden Sie auf der Seite *Configure Liveness, Readiness and Startup Probes* der Kubernetes-Website unter

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/>

► Angeleitete Übung

Aktivieren von Probes

In dieser Übung konfigurieren Sie Liveness- und Readiness-Probes, um den Zustand einer in Ihrem Red Hat OpenShift-Cluster bereitgestellten Anwendung zu überwachen.

Die von Ihnen in dieser Übung bereitgestellte Anwendung stellt zwei HTTP GET-Endpunkte zur Verfügung:

- Der Endpunkt `/healthz` antwortet mit dem HTTP-Statuscode `200`, wenn der Anwendungs-Pod Anforderungen empfangen kann.

Der Endpunkt gibt an, dass der Anwendungs-Pod fehlerfrei und erreichbar ist. Er gibt nicht an, dass die Anwendung bereit ist, Anforderungen zu verarbeiten.

- Der Endpunkt `/ready` antwortet mit einem HTTP-Statuscode von `200`, wenn die Gesamtanwendung funktioniert.

Der Endpunkt gibt an, dass die Anwendung bereit ist, Anforderungen zu verarbeiten.

In dieser Übung antwortet der Endpunkt `/ready` mit dem HTTP-Statuscode `200`, wenn der Anwendungs-Pod gestartet wird. Der Endpunkt `/ready` antwortet in den ersten 30 Sekunden nach der Bereitstellung mit dem HTTP-Statuscode `503`, um einen langsamen Anwendungsstart zu simulieren.

Sie konfigurieren den Endpunkt `/healthz` für den Liveness-Probe und den Endpunkt `/ready` für den Readiness-Test.

Sie simulieren Netzwerkausfälle in Ihrem Red Hat OpenShift-Cluster und beobachten das Verhalten in den folgenden Szenarien:

- Die Anwendung ist nicht verfügbar.
- Die Anwendung ist verfügbar, kann aber nicht auf die Datenbank zugreifen. Daher kann sie keine Anforderungen verarbeiten.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Konfigurieren der Readiness- und Liveness-Probes für eine Anwendung über die Befehlszeile
- Suchen nach Probe-Fehlermeldungen im Ereignis-Log

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen ausgeführten Red Hat OpenShift-Cluster
- Auf das S2I-Builder-Image für Node.js
- Auf die Beispielanwendung im Git-Repository D0288-apps (Probes).

Führen Sie den folgenden Befehl auf `workstation` aus, um die Voraussetzungen für die Übung zu überprüfen und die Lab-Dateien herunterzuladen:

```
[student@workstation ~]$ lab probes start
```

Anweisungen

- 1. Erstellen Sie ein neues Projekt, und stellen Sie anschließend die im Unterverzeichnis `probes` des Git-Repositorys enthaltene Beispielanwendung auf einem Red Hat OpenShift-Cluster bereit.

- 1.1. Führen Sie mit dem Befehl „source“ die Konfiguration Ihrer Kursumgebung aus:

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

- 1.2. Melden Sie sich bei Red Hat OpenShift mit Ihrem Entwicklerbenutzerkonto an:

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
```

- 1.3. Erstellen Sie ein neues Projekt:

```
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-probes
```

- 1.4. Erstellen Sie eine neue Bereitstellung mit folgenden Parametern:

- Name: `probes`
- Build-Image: `nodejs:12`
- Anwendungsverzeichnis: `probes`
- Build-Variablen:
 - Name: `npm_config_registry`, Wert: `http://${RHT_OCP4_NEXUS_SERVER}/repository/nodejs`

Sie können den folgenden Befehl kopieren oder mit dem Skript `/home/student/D0288/labs/probes/oc-new-app.sh` ausführen:

```
[student@workstation ~]$ oc new-app \
--name probes --context-dir probes --build-env \
npm_config_registry=http://${RHT_OCP4_NEXUS_SERVER}/repository/nodejs \
nodejs:12~https://github.com/${RHT_OCP4_GITHUB_USER}/D0288-apps
--> Success
...output omitted...
```

Beachten Sie, dass vor und hinter dem Gleichheitszeichen (=) nach `npm_config_registry` kein Leerzeichen steht.

- 1.5. Sehen Sie sich die Build-Logs an. Warten Sie, bis der Build abgeschlossen ist und das Anwendungscontainer-Image an die Red Hat OpenShift-Registry übertragen wurde:

```
[student@workstation ~]$ oc logs -f bc/probes  
...output omitted...  
Push successful
```

- 1.6. Warten Sie, bis die Anwendung bereitgestellt wird. Zeigen Sie den Status des Anwendungs-Pods an. Der Anwendungs-Pod sollte den Status *Running* aufweisen:

```
[student@workstation ~]$ oc get pods  
NAME          READY   STATUS    RESTARTS   AGE  
probes-1-build   0/1     Completed   0          46s  
probes-6cc59f8f98-vxfw9   1/1     Running    0          10s
```

- 2. Testen Sie die Endpunkte */ready* und */healthz* der Anwendung manuell.

- 2.1. Verwenden Sie eine Route, um die Anwendung für den externen Zugriff bereitzustellen:

```
[student@workstation ~]$ oc expose svc probes  
route.route.openshift.io/probes exposed
```

- 2.2. Testen Sie den Endpunkt */ready*:

```
[student@workstation ~]$ curl \  
-i probes-$RHT_OCP4_DEV_USER}-probes.${RHT_OCP4_WILDCARD_DOMAIN}/ready
```

Der Endpunkt */ready* simuliert einen langsamen Start der Anwendung. Für die ersten 30 Sekunden nach dem Start der Anwendung wird der HTTP-Statuscode 503 mit der folgenden Antwort zurückgegeben:

```
HTTP/1.1 503 Service Unavailable  
...output omitted...  
Error! Service not ready for requests...
```

Nachdem die Anwendung für 30 Sekunden ausgeführt wurde, lautet die Antwort:

```
HTTP/1.1 200 OK  
...output omitted...  
Ready for service requests...
```

- 2.3. Testen Sie den Endpunkt */healthz* der Anwendung:

```
[student@workstation ~]$ curl \  
-i probes-$RHT_OCP4_DEV_USER}-probes.${RHT_OCP4_WILDCARD_DOMAIN}/healthz  
HTTP/1.1 200 OK  
...output omitted...  
OK
```

- 2.4. Testen Sie die Anwendungsantwort:

```
[student@workstation ~]$ curl \
probes-${RHT_OCP4_DEV_USER}-probes.${RHT_OCP4_WILDCARD_DOMAIN}
Hello! This is the index page for the app.
```

► 3. Aktivieren Sie die Readiness- und Liveness-Probes für die Anwendung.

- 3.1. Führen Sie den Befehl `oc set` aus, um die Liveness- und Readiness-Probes mit den folgenden Parametern zu konfigurieren:
 - Verwenden Sie für den Liveness-Probe den Endpunkt `/healthz` auf Port 8080.
 - Verwenden Sie für den Readiness-Probe den Endpunkt `/ready` auf Port 8080.
 - Für beide Probes :
 - Konfigurieren Sie eine Anfangsverzögerung von 2 Sekunden.
 - Konfigurieren Sie den Timeout von 2 Sekunden.

```
[student@workstation ~]$ oc set probe deployment probes --liveness \
--get-url=http://:8080/healthz \
--initial-delay-seconds=2 --timeout-seconds=2
deployment.apps/probes probes updated
[student@workstation ~]$ oc set probe deployment probes --readiness \
--get-url=http://:8080/ready \
--initial-delay-seconds=2 --timeout-seconds=2
deployment.apps/probes probes updated
```

- 3.2. Verifizieren Sie den Wert in den Einträgen `livenessProbe` und `readinessProbe`:

```
[student@workstation D0288-apps]$ oc describe deployment probes | \
grep -iA 1 liveness
  Liveness: http-get http://:8080/healthz delay=2s timeout=2s period=10s
#success=1 #failure=3
  Readiness: http-get http://:8080/ready delay=2s timeout=2s period=10s
#success=1 #failure=3
```

- 3.3. Warten Sie, bis der Anwendungs-Pod erneut bereitgestellt wird und in den Status `READY` wechselt:

```
[student@workstation D0288-apps]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
...output omitted...
probes-6cc59f8f98-fsf8x8  0/1   Running     0          6s
```

Vom Status `READY` wird `0/1` angezeigt, falls der Wert `AGE` kleiner als etwa 30 Sekunden ist. Der Status `READY` lautet anschließend `1/1`:

```
[student@workstation D0288-apps]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
...output omitted...
probes-6cc59f8f98-fsf8x8  1/1   Running     0          62s
```

Kapitel 7 | Verwalten der Anwendungsbereitstellungen

- 3.4. Führen Sie den Befehl `oc logs` aus, um die Ergebnisse der Liveness- und Readiness-Probes anzuzeigen:

```
[student@workstation ~]$ POD=$(oc get pods -o name | grep -v build)
[student@workstation ~]$ oc logs -f $POD
...output omitted...
nodejs server running on http://0.0.0.0:8080
ping /healthz => pong [healthy]
ping /ready => pong [notready]
ping /healthz => pong [healthy]
ping /ready => pong [notready]
ping /healthz => pong [healthy]
ping /ready => pong [ready]
...output omitted...
```

Beobachten Sie, dass der Readiness-Probe für etwa 30 Sekunden nach der erneuten Bereitstellung fehlschlägt und anschließend erfolgreich ist. Beachten Sie, dass die Anwendung eine langsame Initialisierung der Anwendung simuliert, indem die Einstellung einer 30-sekündigen Verzögerung erzwungen wird, bevor sie mit dem Status "ready" antwortet.

Beenden Sie diesen Befehl nicht. Im nächsten Schritt überwachen Sie die Ausgabe dieses Befehls weiter.

► 4. Simulieren Sie einen Netzwerkfehler.

Bei einem Netzwerkausfall reagiert ein Service nicht mehr. Dies bedeutet, dass die Liveness- und Readiness-Probes fehlschlagen.

Red Hat OpenShift kann das Problem beheben, indem der Container auf einem anderen Knoten neu erstellt wird.

- 4.1. Führen Sie in einem anderen Terminalfenster oder auf einem anderen Tab das Skript `~/D0288/labs/probes/kill.sh` aus, um einen Fehler beim Liveness-Probe zu simulieren:

```
[student@workstation ~]$ ~/D0288/labs/probes/kill.sh
Switched app state to unhealthy...
Switched app state to not ready...
```

- 4.2. Kehren Sie zum Terminal zurück, an dem Sie die Anwendungsbereitstellung überwachen:

```
[student@workstation ~]$ oc logs -f $POD
...output omitted...
ping /healthz => pong [healthy]
Received kill request for health probe.
Received kill request for readiness probe.
ping /ready => pong [notready]
ping /healthz => pong [unhealthy]
ping /ready => pong [notready]
ping /healthz => pong [unhealthy]
ping /ready => pong [notready]
ping /healthz => pong [unhealthy]
```

```
npm info lifecycle probes@1.0.0~poststart: probes@1.0.0
npm timing npm Completed in 150591ms
npm info ok
```

Red Hat OpenShift startet den Pod neu, wenn der Liveness-Probe wiederholt fehlschlägt (standardmäßig drei fehlgeschlagene Probes). Dies bedeutet, dass Red Hat OpenShift die Anwendung auf einem verfügbaren Knoten, der nicht von dem Netzwerkfehler betroffen ist, neu startet.

Diese Logausgabe wird nur angezeigt, wenn Sie die AnwendungsLogs sofort überprüfen, nachdem Sie die Beendigungsanforderung ausgeführt haben. Wenn Sie die Logs überprüfen, nachdem Red Hat OpenShift den Pod neu gestartet hat, sind die Logs bereinigt, und Sie sehen nur die im nächsten Schritt angezeigte Ausgabe.

- 4.3. Verifizieren Sie, dass Red Hat OpenShift den fehlerhaften Pod neu startet. Überprüfen Sie weiterhin die Ausgabe des Befehls `oc get pods`. Beobachten Sie die Spalte RESTARTS, und verifizieren Sie, dass die Anzahl größer als 0 ist:

```
[student@workstation D0288-apps]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
...output omitted...
probes-6cc59f8f98-fsf8x8  1/1     Running   1          62s
```

Warten Sie, bis der vorherige Pod beendet wird, bevor Sie mit dem nächsten Schritt fortfahren.

- 4.4. Prüfen Sie die AnwendungsLogs. Der Liveness-Probe wird erfolgreich ausgeführt und die Anwendung meldet einen fehlerfreien Zustand.

```
[student@workstation ~]$ POD=$(oc get pods -o name | grep -v build)
[student@workstation ~]$ oc logs -f $POD
...output omitted...
ping /ready => pong [ready]
ping /healthz => pong [healthy]
...output omitted...
```

- 5. Simulieren Sie einen Fehler beim Erreichen der Anwendungsdatenbank.

Bei einem Netzwerkfehler zwischen der Anwendung und der Datenbank antwortet die Anwendung weiterhin auf Anforderungen, kann jedoch keine Anforderungen verarbeiten.

Dies bedeutet, dass der Liveness-Probe erfolgreich ausgeführt wird, der Readiness-Test jedoch fehlschlägt.

- 5.1. Führen Sie das Skript `~/D0288/labs/probes/not-ready.sh` aus, um einen Fehler beim Readiness-Probe zu simulieren:

```
[student@workstation ~]$ ~/D0288/labs/probes/not-ready.sh
Switched app state to not ready...
```

- 5.2. Prüfen Sie die AnwendungsLogs. Der Readiness-Probe gibt einen Fehler zurück.

Kapitel 7 | Verwalten der Anwendungsbereitstellungen

```
[student@workstation ~]$ oc logs -f $POD
...output omitted...
ping /ready => pong [notready]
ping /healthz => pong [healthy]
...output omitted...
```

- 5.3. Überprüfen Sie, ob sich der Anwendungs-Pod nicht im Status READY befindet:

```
[student@workstation DO288-apps]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
...output omitted...
probes-6cc59f8f98-fsf8x  0/1     Running   1          78m
```

- 5.4. Überprüfen Sie, ob der Service nicht verfügbar ist:

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
[student@workstation ~]$ curl -is \
probes-${RHT_OCP4_DEV_USER}-probes.${RHT_OCP4_WILDCARD_DOMAIN} \
| grep 'HTTP/1.0'

HTTP/1.0 503 Service Unavailable
```

In einer Produktionsumgebung würde Red Hat OpenShift Anforderungen an redundante Anwendungs-Pods umleiten. Wenn die Anwendung die Datenbank erreicht, wird der Readiness-Probe erneut erfolgreich ausgeführt, der Pod wechselt in den Status READY, und Red Hat OpenShift setzt das Weiterleiten des Datenverkehrs an den Pod fort.

- 6. Verifizieren Sie, dass sie das Fehlschlagen der Probes im EreignisLog sehen können.

Verwenden Sie den Befehl `oc describe` auf dem Pod aus dem vorherigen Schritt:

```
[student@workstation ~]$ POD=$(oc get pods -o name | grep -v build)
[student@workstation ~]$ oc describe $POD
Events:
  Type      Reason     Age   From      Message
  ----      ----     --   --       --
...output omitted...
  Warning   Unhealthy  ...   ...     Liveness probe failed: ... statuscode: 503
  Normal    Killing    ...   ...     Container probes failed liveness probe, will be
  restarted
...output omitted...
  Warning   Unhealthy  ...   ...     Readiness probe failed: ... statuscode: 503
```

- 7. Führen Sie eine Bereinigung durch. Löschen Sie das Projekt `probes` in Red Hat OpenShift:

```
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-probes
```

Beenden

Führen Sie auf der workstation den Befehl `lab probes finish` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken.

```
[student@workstation ~]$ lab probes finish
```

Hiermit ist die angeleitete Übung beendet.

Auswählen der geeigneten Deploymentstrategie

Ziele

In diesem Abschnitt wird die Auswahl der geeigneten Deploymentstrategie für eine Cloud-native Anwendung beschrieben.

Auswählen von DeploymentConfig anstatt Deployments

Standardmäßig erstellen die meisten Red Hat OpenShift Container Platform-Befehle, beispielsweise `oc new-app`, Deployment-Ressourcen. Dabei handelt es sich um native First-Class-API-Ressourcen, die in jeder Kubernetes-Distribution enthalten sind. Red Hat OpenShift stellt jedoch eine Ressource mit dem Namen DeploymentConfig bereit.

Mit der DeploymentConfig-Ressource können Sie zusätzliche Funktionen verwenden, z. B.

- Benutzerdefinierte Deploymentstrategien
- Lifecycle-Hooks

Die Deployment-Ressource unterstützt keine benutzerdefinierten Strategien.

Mit der Deployment-Ressource können Sie Container-Lifecycle-Hooks wie `PostStart` und `PreStop` definieren, die den Lifecycle-Hooks ähneln, die von der DeploymentConfig-Ressource bereitgestellt werden. Es wird jedoch nicht garantiert, dass die Deployment-Container-Lifecycle-Hooks vor dem Befehl `ENTRYPOINT` des Container-Pods ausgeführt werden.

Folglich schränkt dies den Einsatz des Deployment-Container-Lifecycle-Hooks ein.

Verwenden Sie die Deployment-Ressource, wenn Sie keine zusätzlichen Features benötigen, die von der DeploymentConfig-Ressource bereitgestellt werden, oder wenn Ihre Bereitstellungen mit anderen Distributionen von Kubernetes kompatibel sein müssen.

Deploymentstrategien in Red Hat OpenShift

Eine Deploymentstrategie ist eine Methode zur Änderung oder Aktualisierung einer Anwendung. Das Ziel besteht darin, Änderungen oder Upgrades bei minimaler Ausfallzeit und mit einer reduzierten Auswirkung auf Endbenutzer vorzunehmen.

Red Hat OpenShift stellt verschiedene Deploymentstrategien bereit. Diese Strategien können in zwei Hauptkategorien unterteilt werden:

- Durch Verwendung der in der AnwendungsDeploymentkonfiguration definierten Deploymentstrategie.
- Durch Verwendung des Red Hat OpenShift-Routers, um den Datenverkehr an bestimmte Anwendungs-Pods weiterzuleiten.

Die in der Deploymentkonfiguration definierten Strategien wirken sich auf alle Routen aus, welche die Anwendung verwenden. Strategien, die Router-Features verwenden, wirken sich auf einzelne Routen aus.

Strategien, die das Ändern der Deploymentkonfiguration umfassen, werden im Folgenden aufgeführt:

Rolling

Die Rolling-Strategie ist die Standardstrategie.

Diese Strategie ersetzt schrittweise Instanzen der vorherigen Version einer Anwendung durch Instanzen der neuen Version der Anwendung. Diese Strategie führt den Readiness-Probe aus, um zu bestimmen, wann ein neuer Pod bereit ist. Nachdem ein Readiness-Probe für den neuen Pod erfolgreich ausgeführt wurde, skaliert der Deployment-Controller den alten Pod herunter.

Falls ein Problem auftritt, bricht der Deployment-Controller die Rolling-Bereitstellung ab. Entwickler können die Rolling-Bereitstellung auch manuell mit dem Befehl `oc rollout cancel` abbrechen.

Rolling-Bereitstellungen in Red Hat OpenShift sind Canary-Bereitstellungen. Red Hat OpenShift testet eine neue Version (der Canary), bevor alle alten Instanzen ersetzt werden. Falls der Readiness-Probe niemals erfolgreich ist, entfernt Red Hat OpenShift die Canary-Instanz und setzt die Deploymentkonfiguration automatisch zurück.

Verwenden Sie eine Rolling-Deploymentstrategie in den folgenden Fällen:

- Während einer Anwendungsaktualisierung soll es keine Ausfallzeit geben.
- Ihre Anwendung unterstützt die gleichzeitige Ausführung einer älteren Version und einer neueren Version.

Abbildung 7.3: Deploymentstrategie für Rolling-Updates

Recreate

Bei dieser Strategie hält Red Hat OpenShift zunächst alle Pods an, die aktuell ausgeführt werden, und startet nur dann Pods mit der neuen Version der Anwendung. Diese Strategie ist von einer Ausfallzeit betroffen, da für einen kurzen Zeitraum keine Instanzen Ihrer Anwendung ausgeführt werden.

Verwenden Sie eine Recreate-Deploymentstrategie in den folgenden Fällen:

Kapitel 7 | Verwalten der Anwendungsbereitstellungen

- Ihre Anwendung unterstützt nicht die gleichzeitige Ausführung einer älteren Version und einer neueren Version.
- Ihre Anwendung verwendet ein persistentes Volume mit dem RWO-Zugriffsmodus (ReadWriteOnce), was Schreibvorgänge aus mehreren Pods nicht zulässt.

Custom

Wenn weder die Rolling- noch die Recreate-Deploymentstrategien Ihren Anforderungen gerecht werden, können Sie die benutzerdefinierte Deploymentstrategie verwenden, um Ihre Anwendungen bereitzustellen. Manchmal muss der auszuführende Befehl besser an das System angepasst werden (z. B. Arbeitsspeicher für die Java Virtual Machine), oder Sie benötigen ein benutzerdefiniertes Image mit intern entwickelten Libraries, die der breiten Öffentlichkeit nicht zur Verfügung stehen.

Verwenden Sie für diese Anwendungsfälle die Custom-Strategie. Sie können die Strategie in einer `DeploymentConfig`-Ressource im Attribut `spec.strategy.type` angeben.

Sie können Ihr eigenes benutzerdefiniertes Container-Image bereitstellen, in dem Sie das Deploymentverhalten definieren. Dieses benutzerdefinierte Image wird im Attribut `spec.strategy.customParams.image` der Deploymentkonfiguration der Anwendung definiert. Sie können auch Umgebungsvariablen und den Befehl anpassen, der für die Bereitstellung ausgeführt werden soll.

Integration von Red Hat OpenShift-Bereitstellungen mit Lifecycle-Hooks

Die Recreate- und Rolling-Strategien unterstützen Lifecycle-Hooks. Sie können diese Hooks verwenden, um Ereignisse an vorab definierten Punkten im Deploymentprozess auszulösen. Red Hat OpenShift-Bereitstellungen enthalten drei Lifecycle-Hooks:

Pre-Lifecycle-Hook

Red Hat OpenShift führt den Pre-Lifecycle-Hook aus, bevor neue Pods für eine Bereitstellung gestartet und auch bevor ältere Pods heruntergefahren werden.

Mid-Lifecycle-Hook

Der Mid-Lifecycle-Hook wird ausgeführt, nachdem alle alten Pods in einer Bereitstellung heruntergefahren, jedoch bevor neue Pods gestartet wurden. Mid-Lifecycle-Hooks stehen nur für die Recreate-Strategie zur Verfügung.

Post-Lifecycle-Hook

Der Post-Lifecycle-Hook wird ausgeführt, nachdem alle neuen Pods für eine Bereitstellung gestartet und nachdem alle älteren Pods heruntergefahren wurden.

Diese Lifecycle-Hooks werden in der `Strategy`-Ressource unter einem von drei Attributen definiert:

- `rollingParams` für Rolling-Strategien.
- `recreateParams` für Recreate-Strategien.
- `customParams` für Custom-Strategien.

Sie können das Attribut `pre,mid` und `post` hinzufügen, das die Lifecycle-Hooks enthält.

Lifecycle-Hooks werden in getrennten, kurzfristigen Containern ausgeführt. Nach der Ausführung werden sie durch Red Hat OpenShift automatisch bereinigt. Die automatische Datenbankinitialisierung und Datenbankmigrationen sind typische Anwendungsfälle für Lifecycle-Hooks.

Jeder Hook besitzt das Attribut `failurePolicy`. Dieses legt fest, welche Aktion ausgeführt wird, wenn ein Hook-Fehler erkannt wird. Es gibt drei Richtlinien:

- "Abort": Der Deploymentprozess wird bei einem Hook-Fehler als fehlgeschlagen angesehen.
- "Retry": Die Hook-Ausführung wird so lange erneut versucht, bis sie erfolgreich ist.
- "Ignore": Hook-Fehler werden ignoriert, und die Bereitstellung darf fortgesetzt werden.

Implementieren erweiterter Deploymentstrategien mithilfe des Red Hat OpenShift-Routers

Erweiterte Deploymentstrategien, welche die Red Hat OpenShift-Router-Features verwenden, werden im Folgenden aufgeführt:

Blue-Green-Bereitstellung

In Blue-Green-Bereitstellungen haben Sie zwei identische, gleichzeitig ausgeführte Umgebungen, wobei jede Umgebung eine andere Version der Anwendung ausführt.

Der Red Hat OpenShift-Router wird verwendet, um den Datenverkehr von der aktuellen, in der Produktion befindlichen Version (grün) zur neueren, aktualisierten Version (blau) zu leiten. Sie können diese Strategie mithilfe einer Route und zweier Services implementieren. Definieren Sie einen Service für jede bestimmte Version der Anwendung.

Die Route verweist auf einen der Services zu einem bestimmten Zeitpunkt und kann so geändert werden, dass sie auf einen anderen Service verweist, wenn dieser bereit ist, oder dass sie einen Rollback ausführt. Als Entwickler können Sie die neue Version Ihrer Anwendung testen. Stellen Sie dazu eine Verbindung zum neuen Service her, bevor Sie Ihren Produktionsdatenverkehr zu ihr weiterleiten. Wenn Ihre neue Anwendungsversion für die Produktion bereit ist, ändern Sie den Produktions-Router so, dass er auf den neuen Service verweist, der für Ihre aktualisierte Anwendung definiert ist.

A/B-Bereitstellung

Bei der A/B-Bereitstellung können Sie eine neue Version der Anwendung für einen begrenzten Satz an Benutzern in der Produktionsumgebung bereitstellen. Sie können Red Hat OpenShift so konfigurieren, dass die Mehrzahl der Anforderungen an die aktuell bereitgestellte Version in einer Produktionsumgebung gesendet wird, während eine begrenzte Anzahl an Anforderungen an die neue Version gesendet wird.

Durch Steuern des Teils der Anforderungen, die im Verlauf des Tests an jede Version gesendet werden, können Sie die Anzahl der an die neue Version gesendeten Anforderungen schrittweise erhöhen. Schließlich können Sie dafür sorgen, dass der Datenverkehr nicht mehr an die vorherige Version weitergeleitet wird. Während Sie die Anforderungslast für jede Version anpassen, muss die Anzahl der Pods im jeweiligen Service möglicherweise skaliert werden, um die entsprechende Leistung bereitzustellen.

Berücksichtigen Sie außerdem die N-1-Kompatibilität und die kontrollierte Beendigung:

N-1-Kompatibilität

Für viele Deploymentstrategien müssen zwei Versionen der Anwendung gleichzeitig ausgeführt werden. Wenn Sie zwei Versionen einer Anwendung gleichzeitig ausführen, stellen Sie sicher, dass die vom neuen Code geschriebenen Daten von der alten Version des Codes gelesen und behandelt (oder ordnungsgemäß ignoriert) werden können. Dies wird als N-1-Kompatibilität bezeichnet.

Kapitel 7 | Verwalten der Anwendungsbereitstellungen

Die von der Anwendung verwalteten Daten können viele Formen annehmen: Auf der Disk, in einer Datenbank oder in einem temporären Cache gespeicherte Daten. Die meisten, gut konzipierten zustandslosen Webanwendungen können Rolling-Bereitstellungen unterstützen. Es ist jedoch wichtig, die Anwendung dahingehend zu testen und zu konzipieren, dass sie die N-1-Kompatibilität verarbeiten kann.

Kontrollierte Beendigung

Red Hat OpenShift ermöglicht, dass Anwendungsinstanzen kontrolliert heruntergefahren werden können, bevor die Instanzen aus der Lastverteilungsliste des Routers entfernt werden. Anwendungen müssen gewährleisten, dass sie Benutzerverbindungen kontrolliert beenden, bevor sie beendet werden.

Red Hat OpenShift sendet ein SIGTERM-Signal an die Prozesse im Container, wenn er heruntergefahren werden soll. Der Anwendungscode sollte beim Empfang eines SIGTERM-Signals aufhören, neue Verbindungen zu akzeptieren. Durch das Stoppen neuer Verbindungen wird gewährleistet, dass der Router Datenverkehr an die anderen aktiven Instanzen weiterleiten kann. Der Anwendungscode sollte anschließend warten, bis alle offenen Verbindungen geschlossen sind (oder die einzelnen Verbindungen bei der nächsten Gelegenheit kontrolliert beenden), bevor der Vorgang beendet wird.

Nachdem der Zeitraum der kontrollierten Beendigung abgelaufen ist, sendet Red Hat OpenShift ein SIGKILL-Signal an die nicht beendeten Prozesse. Dadurch wird der jeweilige Prozess sofort beendet. Das Attribut `terminationGracePeriodSeconds` eines Pods oder einer Pod-Vorlage steuert den Zeitraum der kontrollierten Beendigung (die Standardeinstellung ist 30 Sekunden) und kann je nach Anwendung angepasst werden.



Literaturhinweise

Weitere Informationen zu Deploymentstrategien finden Sie im Kapitel *Deployments* im Handbuch *Applications* für Red Hat OpenShift Container Platform 4.6 unter https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/applications/index#deployments

Eine ausführliche Erläuterung der Unterschiede zwischen Deployment- und DeploymentConfig-Ressourcen finden Sie unter <https://docs.openshift.com/container-platform/4.6/applications/deployments/what-deployments-are.html>

Eine Einführung in Blue-Green-, Canary- und Rolling-Bereitstellungen

<https://opensource.com/article/17/5/colorful-deployments>

Strategien für den Anwendungs-Release mit OpenShift

<https://access.redhat.com/articles/2897391>

Eine Beschreibung der für Deployment-Ressourcen verfügbaren Lifecycle-Hooks finden Sie unter

Container-Lifecycle-Hooks

<https://kubernetes.io/docs/concepts/containers/container-lifecycle-hooks/>

► Angeleitete Übung

Implementieren einer Deploymentstrategie

In dieser Übung initialisieren Sie eine Datenbank mit dem Lifecycle-Hook einer Bereitstellung.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Ändern der Deploymentstrategie einer MySQL-Datenbank-DeploymentConfig-Ressource in Recreate
- Hinzufügen eines Lifecycle-Hooks nach der Bereitstellung zur DeploymentConfig-Ressource zum Initialisieren der MySQL-Datenbank mit Daten aus einer SQL-Datei
- Diagnostizieren und Beheben von nach der Bereitstellung des Lifecycles auftretenden Ausführungsproblemen

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen ausgeführten Red Hat OpenShift-Cluster
- Auf das MySQL 8.0-Container-Image (`rhel8/mysql-80`)

Führen Sie den folgenden Befehl auf `workstation` aus, um die Voraussetzungen für die Übung zu überprüfen und die Lab- und Lösungsdateien herunterzuladen:

```
[student@workstation ~]$ lab strategy start
```

Anweisungen

- 1. Erstellen Sie ein neues Projekt, und stellen Sie eine Anwendung anhand des Container-Images `rhel8/mysql-80` im Red Hat OpenShift-Cluster bereit.
- 1.1. Führen Sie mit dem Befehl „source“ die Konfiguration Ihrer Kursumgebung aus:

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

- 1.2. Melden Sie sich bei Red Hat OpenShift mit Ihrem Entwicklerbenutzernamen an:

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

Kapitel 7 | Verwalten der Anwendungsbereitstellungen

- 1.3. Erstellen Sie ein neues Projekt für die Anwendung. Stellen Sie dem Projektnamen Ihren Entwicklerbenutzernamen voran.

```
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-strategy
Now using project "youruser-strategy" on server "https://api.cluster.domain.example.com:6443".
...output omitted...
```

- 1.4. Verwenden Sie den Befehl `oc new-app`, um eine neue Anwendung mit den folgenden Parametern zu erstellen:

- Name: `mysql`
- Variablen:
 - Name: `MYSQL_USER`, Wert: `test`
 - Name: `MYSQL_PASSWORD`, Wert: `redhat`
 - Name: `MYSQL_DATABASE`, Wert: `testdb`
 - Name: `MYSQL_AI0`, Wert: `0`
- Container-Image: `registry.redhat.io/rhel8/mysql-80`
- Verwenden Sie die `deploymentconfig`-Ressource

Kopieren oder führen Sie den folgenden Befehl aus dem Skript `~/D0288/labs/strategy/oc-new-app.sh` aus:

```
[student@workstation ~]$ oc new-app --as-deployment-config \
--name mysql -e MYSQL_USER=test -e MYSQL_PASSWORD=redhat \
-e MYSQL_DATABASE=testdb -e MYSQL_AI0=0 \
--docker-image registry.redhat.io/rhel8/mysql-80
--> Found container image ... "registry.redhat.io/rhel8/mysql-80"
...output omitted...
--> Creating resources ...
...output omitted...
--> Success
...output omitted...
```

- 1.5. Warten Sie, bis der MySQL-Pod bereitgestellt ist. Der Pod sollte den Status READY aufweisen:

NAME	READY	STATUS	RESTARTS	AGE
<code>mysql-1-54bhp</code>	<code>1/1</code>	<code>Running</code>	<code>0</code>	<code>11s</code>
<code>mysql-1-deploy</code>	<code>0/1</code>	<code>Completed</code>	<code>0</code>	<code>16s</code>

► 2. Ändern Sie die Deploymentstrategie in Recreate.

- 2.1. Verifizieren Sie, dass die StandardDeploymentstrategie für die MySQL-Anwendung `Rolling` lautet:

```
[student@workstation ~]$ oc describe dc/mysql | grep -i strategy:  
Strategy: Rolling
```

- 2.2. In den folgenden Schritten nehmen Sie verschiedene Änderungen an der Deploymentkonfiguration vor. Deaktivieren Sie Trigger für Konfigurationsänderungen für die Bereitstellung, um zu verhindern, dass nach jeder Änderung eine erneute Bereitstellung vorgenommen wird:

```
[student@workstation ~]$ oc set triggers dc/mysql --from-config --remove  
deploymentconfig.apps.openshift.io/mysql triggers updated
```

- 2.3. Ändern Sie die StandardDeploymentstrategie. Sie können den Befehl aus dem Skript ~/D0288/labs/strategy/recreate.sh kopieren, das Skript ausführen oder den Befehl wie folgt eingeben:

```
[student@workstation ~]$ oc patch dc/mysql --patch \  
'{"spec":{"strategy":{"type":"Recreate"}}}'  
deploymentconfig.apps.openshift.io/mysql patched
```

- 2.4. Entfernen Sie das Attribut `rollingParams` aus der Deploymentkonfiguration. Sie können den Befehl aus dem Skript ~/D0288/labs/strategy/rm-rolling.sh kopieren oder ausführen:

```
[student@workstation ~]$ oc patch dc/mysql --type=json \  
-p='[{"op":"remove", "path": "/spec/strategy/rollingParams"}]'  
deploymentconfig.apps.openshift.io/mysql patched
```

- 3. Fügen Sie einen Post-Lifecycle-Hook hinzu, um die Daten für die MySQL-Datenbank zu initialisieren.

- 3.1. Überprüfen Sie die Datei ~/D0288/labs/strategy/users.sql. Dieses SQL-Skript erstellt eine Tabelle mit dem Namen `users` und fügt drei Datenzeilen ein:

```
CREATE TABLE IF NOT EXISTS users (  
    user_id int(10) unsigned NOT NULL AUTO_INCREMENT,  
    name varchar(100) NOT NULL,  
    email varchar(100) NOT NULL,  
    PRIMARY KEY (user_id) ENGINE=InnoDB DEFAULT CHARSET=utf8;  
  
insert into users(name,email) values ('user1', 'user1@example.com');  
insert into users(name,email) values ('user2', 'user2@example.com');  
insert into users(name,email) values ('user3', 'user3@example.com');
```

- 3.2. Überprüfen Sie das Skript ~/D0288/labs/strategy/import.sh.

Das Skript lädt das vorherige SQL-Skript herunter und führt es aus, um die Datenbank zu initialisieren. Der Post-Lifecycle-Hook lädt das Skript `import.sh` herunter und führt es aus:

```
#!/bin/bash
...output omitted...

echo 'Downloading SQL script that initializes the database...'
curl -s -O https://github.com/RedHatTraining/D0288-apps/releases/download/
OCP-4.1-1/users.sql

echo "Trying $HOOK_RETRIES times, sleeping $HOOK_SLEEP sec between tries:"
while [ "$HOOK_RETRIES" != 0 ]; do

    echo -n 'Checking if MySQL is up...'
    if mysqlshow -h$MYSQL_SERVICE_HOST -u$MYSQL_USER -p$MYSQL_PASSWORD -P3306
$MYSQL_DATABASE &>/dev/null
    then
        echo 'Database is up'
        break
    else
        echo 'Database is down'

        # Sleep to wait for the MySQL pod to be ready
        sleep $HOOK_SLEEP
    fi

    let HOOK_RETRIES=HOOK_RETRIES-1
done

if [ "$HOOK_RETRIES" = 0 ]; then
    echo 'Too many tries, giving up'
    exit 1
fi

# Run the SQL script
if mysql -h$MYSQL_SERVICE_HOST -u$MYSQL_USER -p$MYSQL_PASSWORD -P3306
$MYSQL_DATABASE < /tmp/users.sql
then
    echo 'Database initialized successfully'
else
    echo 'Failed to initialize database'
    exit 2
fi
```

Das Skript versucht, bei den häufigsten HOOK_RETRIES eine Verbindung zur Datenbank herzustellen und ruht HOOK_SLEEP Sekunden zwischen den Versuchen.

Das Skript implementiert Wiederholungen, da der Hook-Pod gleichzeitig mit dem Datenbank-Pod gestartet wird. Folglich muss der Datenbank-Pod bereit sein, bevor der Pod-Hook das SQL-Skript ausführen kann.

- 3.3. Überprüfen Sie das Skript ~/D0288/labs/strategy/post-hook.sh. Das Skript fügt der DeploymentConfig-Ressource einen neuen Post-Lifecycle-Hook hinzu:

```
[student@workstation ~]$ cat ~/D0288/labs/strategy/post-hook.sh
...output omitted...
oc patch dc/mysql --patch \
'{"spec": {"strategy": {"recreateParams": {"post": {"failurePolicy": "Abort", "execNewPod": {"containerName": "mysql", "command": ["/bin/sh", "-c", "curl -L -s https://github.com/RedHatTraining/D0288-apps/releases/download/OCP-4.1-1/import.sh -o /tmp/import.sh&&chmod 755 /tmp/import.sh&&/tmp/import.sh"]}}}}}'
```

Die lokalen Kopien von `import.sh` und `users.sql` werden Ihnen zu Referenzzwecken bereitgestellt. Der Hook lädt die Dateien während des Starts herunter und führt sie aus.

- 3.4. Führen Sie das Skript `~/D0288/labs/strategy/post-hook.sh` aus:

```
[student@workstation ~]$ ~/D0288/labs/strategy/post-hook.sh
deploymentconfig.apps.openshift.io/mysql patched
```

- ▶ 4. Verifizieren Sie die gepatchte Deploymentkonfiguration, und führen Sie die neue Deploymentkonfiguration ein.
- 4.1. Verifizieren Sie, dass die Deploymentstrategie nun Recreate lautet und dass ein Post-Lifecycle-Hook vorliegt, der das Skript `import.sh` ausführt:

```
[student@workstation ~]$ oc describe dc/mysql | grep -iA 3 'strategy:'
Strategy: Recreate
Post-deployment hook (pod type, failure policy: Abort):
  Container: mysql
  Command: /bin/sh -c curl -L -s ...
```

- 4.2. Erzwingen Sie eine neue Bereitstellung, um die Änderungen an der Strategie und den neuen Post-Lifecycle-Hook zu erzwingen:

```
[student@workstation ~]$ oc rollout latest dc/mysql
deploymentconfig.apps.openshift.io/mysql rolled out
```

- 4.3. Überprüfen Sie, ob ein neuer MySQL-Pod den Status Running erreicht. Anschließend erreichen die Pods `mysql-2-deploy` und `mysql-2-hook-post` den Status Running:

```
[student@workstation ~]$ watch -n 2 oc get pods
NAME          READY   STATUS    RESTARTS   ...   NODE
mysql-2-deploy 1/1     Running   0          ...
mysql-2-hook-post 1/1     Running   0          ...
mysql-2-kbnpr   1/1     Running   0          ...
```

Nach ein paar Sekunden schlagen der Post-Lifecycle-Hook-Pod und der zweite Deployment-Pod fehl:

NAME	READY	STATUS	RESTARTS	AGE
mysql-1-deploy	0/1	Completed	0	13m
mysql-1-vnq68	1/1	Running	0	114s
mysql-2-deploy	0/1	Error	0	2m11s
mysql-2-hook-post	0/1	Error	0	119s

Wenn der Hook und die Deployment-Pods einen fehlerhaften Status erreichen, drücken Sie auf Strg+C, um den Befehl `watch` zu beenden.

Der Pod `mysql-1-vnq68` ist ein neuer Pod. Die zweite Bereitstellung hat den ursprünglichen Pod aus der ersten Bereitstellung beendet. Nach dem Fehlschlagen der zweiten Bereitstellung wurde mithilfe der ursprünglichen Deploymentkonfiguration aus der ersten Bereitstellung ein neuer Pod erstellt.

► 5. Diagnostizieren und beheben Sie den Post-Lifecycle-Hook-Fehler.

- Zeigen Sie die Logs des fehlgeschlagenen Pods an. In den Logs wird gezeigt, dass das Skript versucht, nur einmal eine Verbindung zur Datenbank herzustellen, und dann einen Fehlerstatus zurückgibt:

```
[student@workstation ~]$ oc logs mysql-2-hook-post
Downloading SQL script that initializes the database...
Trying 0 times, sleeping 2 sec between tries:
Too many tries, giving up
```

- Beachten Sie, dass das Skript fälschlicherweise den Wert „0“ für die Variable `HOOK_RETRIES` aufweist, wodurch es nie eine Verbindung zur Datenbank herstellen kann. Erhöhen Sie Anzahl, wie oft das Skript versuchen soll, eine Verbindung zum Datenbankserver herzustellen. Legen Sie die Umgebungsvariable `HOOK_RETRIES` in der Deploymentkonfiguration auf den Wert 5 fest.

```
[student@workstation ~]$ oc set env dc/mysql HOOK_RETRIES=5
deploymentconfig.apps.openshift.io/mysql updated
```

- Starten Sie unter Verwendung der neuen Werte für die Umgebungsvariablen eine dritte Bereitstellung, um den Hook ein zweites Mal auszuführen:

```
[student@workstation ~]$ oc rollout latest dc/mysql
deploymentconfig.apps.openshift.io/mysql rolled out
```

- Warten Sie, bis der neue Post-Lifecycle-Hook-Pod den Status `Completed` aufweist.

```
[student@workstation ~]$ watch -n 2 oc get pods
NAME          READY   STATUS    RESTARTS   ...
mysql-1-deploy 0/1     Completed  0          5m26s
mysql-2-deploy 0/1     Error     0          3m30s
mysql-2-hook-post 0/1     Error     0          3m8s
mysql-3-29jwt 1/1     Running   0          57s
mysql-3-deploy 0/1     Completed  0          79s
mysql-3-hook-post 0/1     Completed  0          48s
```

- 5.5. Öffnen Sie ein neues Terminal, um die Logs des aktiven Post-Lifecycle-Hook-Pods anzuzeigen. Darin wird gezeigt, dass das Skript ein paar Mal eine Verbindung zur Datenbank herstellen kann und anschließend einen erfolgreichen Status zurückgibt:

```
[student@workstation ~]$ oc logs -f mysql-3-hook-post
Downloading SQL script that initializes the database...
Trying 5 times, sleeping 2 sec between tries:
Checking if MySQL is up...Database is up
mysql: [Warning] Using a password on the command line interface can be insecure.
Database initialized successfully
```

Die Anzahl der Versuche hängt von Ihrer Hardware und den Knoten ab, die der jeweilige Pod entsprechend der Planung durch Red Hat OpenShift ausführen soll.

Wenn der Hook das erste Mal verbunden ist, wird der Pod beendet, wenn Sie versuchen, seine Logs anzuzeigen. Sie können zum nächsten Schritt wechseln.

- 5.6. Nach einigen Sekunden wird der neue Datenbank-Pod unter Berücksichtigung der neuesten Änderungen an der Deploymentkonfiguration erstellt:

NAME	READY	STATUS	RESTARTS	AGE
mysql-1-deploy	0/1	Completed	0	4m29s
mysql-2-deploy	0/1	Error	0	2m2s
mysql-2-hook-post	0/1	Error	0	113s
mysql-3-deploy	0/1	Completed	0	62s
mysql-3-29jwt	1/1	Running	0	49s
mysql-3-hook-post	0/1	Completed	0	45s

Drücken Sie Strg+C nach der Ausführung und Beendigung des Pods **mysql-3-hook-post**, um den Befehl `watch` zu beenden.

Beachten Sie, dass Red Hat OpenShift die während der vorherigen Deploymentversuche erstellten fehlerhaften Pods beibehält. Entsprechend können Sie ihre Logs für die Fehlerbehebung anzeigen.

- 6. Verifizieren Sie, dass der neue MySQL-Datenbank-Pod Daten aus der SQL-Datei enthält.

- 6.1. Öffnen Sie eine Shell-Sitzung, und navigieren Sie zum MySQL-Container-Pod. Führen Sie den Befehl `oc get pods` aus, um den Namen des aktuellen MySQL-Pods abzurufen:

```
[student@workstation ~]$ oc get pods
NAME      READY     STATUS    RESTARTS   AGE
...output omitted...
mysql-3-3p4m1  1/1     Running   0          8m
[student@workstation ~]$ oc rsh mysql-3-3p4m1
sh-4.2$
```

- 6.2. Verifizieren Sie, dass die Tabelle `users` erstellt und die Daten aus der SQL-Datei eingetragen wurden:

```
sh-4.2$ mysql -u$MYSQL_USER -p$MYSQL_PASSWORD $MYSQL_DATABASE -e "select * from
users;" 
...output omitted...
+-----+-----+-----+
```

user_id	name	email
1	user1	user1@example.com
2	user2	user2@example.com
3	user3	user3@example.com

6.3. Beenden Sie die MySQL-Sitzung und die Container-Shell:

```
sh-4.2$ exit  
exit
```

- 7. Führen Sie eine Bereinigung durch.

Löschen Sie das Projekt **strategy**:

```
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-strategy
```

Beenden

Führen Sie auf der **workstation** den Befehl **lab strategy finish** aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken.

```
[student@workstation ~]$ lab strategy finish
```

Hiermit ist die angeleitete Übung beendet.

Verwalten von Anwendungsbereitstellungen mit CLI-Befehlen

Ziele

In diesem Abschnitt wird beschrieben, wie die Bereitstellung einer Anwendung mithilfe von CLI-Befehlen verwaltet wird.

Deploymentkonfiguration

Eine Deploymentkonfiguration definiert die Vorlage für einen Pod und verwaltet die Bereitstellung neuer Images oder Konfigurationsänderungen, sobald die Attribute geändert werden.

Deployment-Konfigurationen können viele verschiedene Deployment Patterns unterstützen, darunter den vollständigen Neustart, anpassbare Rolling-Updates sowie Pre- und Post-Lifecycle-Hooks.

Red Hat OpenShift erstellt bei der Erstellung einer Deployment-Konfiguration automatisch einen Replication Controller, der die Pod-Vorlage der Deployment-Konfiguration repräsentiert.

Wenn sich die Deployment-Konfiguration ändert, erstellt Red Hat OpenShift einen neuen Replication Controller mit der neuesten Pod-Vorlage, und ein Deployment-Prozess wird ausgeführt, um den alten Replication Controller herunterzuskalieren und den neuen Replication Controller hochzuskalieren. Darüber hinaus fügt red Hat OpenShift Instanzen der Anwendung automatisch den Service-Load-Balancern und Routern hinzu und entfernt sie von ihnen, wenn sie gestartet oder gestoppt werden.

Eine Deploymentkonfiguration wird in einem `DeploymentConfig`-Attribut in einer Ressourcendatei deklariert, was im YAML- oder JSON-Format erfolgen kann. Führen Sie den Befehl `oc` aus, um die Deploymentkonfiguration wie jede andere Red Hat OpenShift-Ressource zu verwalten. Die folgende Vorlage zeigt eine Deploymentkonfiguration im YAML-Format:

```
kind: "DeploymentConfig"
apiVersion: "v1"
metadata:
  name: "frontend" ①
spec:
  ...
  replicas: 5 ②
  selector:
    name: "frontend"
  triggers:
    - type: "ConfigChange" ③
    - type: "ImageChange" ④
      imageChangeParams:
  ...
  strategy:
    type: "Rolling"
  ...
```

① Der Name der Deploymentkonfiguration.

Kapitel 7 | Verwalten der Anwendungsbereitstellungen

- ❷ Die Anzahl der auszuführenden Replikate.
- ❸ Ein Trigger für Konfigurationsänderungen, der bewirkt, dass ein neuer Replication Controller erstellt wird, wenn Änderungen an der Deployment-Konfiguration vorgenommen werden.
- ❹ Ein Trigger für Image-Änderungen, der bewirkt, dass immer dann ein neuer Replication Controller erstellt wird, wenn eine neue Image-Version verfügbar ist.

Verwalten von Bereitstellungen mithilfe von CLI-Befehlen

Es gibt verschiedene Befehlszeilenoptionen zum Verwalten von Bereitstellungen. In der folgenden Liste werden die verfügbaren Optionen beschrieben:

- Verwenden Sie den Befehl `oc rollout`, um eine Bereitstellung zu starten. Die Option `latest` legt fest, dass die neueste Version der Vorlage verwendet werden muss:

```
[user@host ~]$ oc rollout latest dc/name
```

Diese Option wird häufig verwendet, um eine neue Bereitstellung zu starten oder eine Anwendung auf die neueste Version zu aktualisieren.

- Führen Sie zum Anzeigen des Verlaufs von Bereitstellungen für eine spezifische Deploymentkonfiguration den Befehl `oc rollout history` aus:

```
[user@host ~]$ oc rollout history dc/name
```

- Hängen Sie für den Zugriff auf eine spezifische Bereitstellung den Parameter `--revision` an den Befehl `oc rollout history` an:

```
[user@host ~]$ oc rollout history dc/name --revision=1
```

- Führen Sie für den Zugriff auf eine Deploymentkonfiguration und deren neueste Revision den Befehl `oc describe dc` aus:

```
[user@host ~]$ oc describe dc name
```

- Führen Sie zum Abbrechen einer Bereitstellung den Befehl `oc rollout` mit der Option `cancel` aus:

```
[user@host ~]$ oc rollout cancel dc/name
```

Sie können eine Bereitstellung abbrechen, wenn der Start zu lange dauert, in der Logdatei Inkonsistenzen bestehen oder die Bereitstellung das Verhalten anderer Ressourcen im System beeinträchtigt.



Warnung

Beim Abbruch wird die Deployment-Konfiguration automatisch auf den zuvor aktiven Replication Controller zurückgesetzt.

- Führen Sie zum Wiederholen einer fehlgeschlagenen Deploymentkonfiguration den Befehl `oc rollout retry dc/name`

```
[user@host ~]$ oc rollout retry dc/name
```

Sie können eine Deploymentkonfiguration wiederholen, nachdem Sie diese Bereitstellung zuvor abgebrochen oder einen Fehler gefunden haben, der zum Fehlschlagen der Bereitstellung führt, wenn Sie dieselbe Revision beibehalten möchten.



Anmerkung

Wenn Sie versuchen, eine Deploymentkonfiguration zu wiederholen, wird der Deploymentprozess gestartet. Es wird jedoch keine neue Deploymentrevision erstellt. Red Hat OpenShift startet den Replication Controller auch mit derselben Konfiguration wie beim Fehlschlagen neu.

- Wenn Sie eine vorherige Version der Anwendung verwenden möchten, können Sie die Bereitstellung durch Ausführen des Befehls `oc rollback` zurücksetzen.

```
[user@host ~]$ oc rollback dc/name
```

Wenn ein Problem mit der neuesten Deploymentkonfiguration besteht, z. B. Benutzer, die sich über ein neues Feature beschweren, das nicht wie erwartet funktioniert, können Sie durch Ausführen des Befehls `oc rollback` die Anwendung auf eine vorherige bekannterenmaßen funktionierende Version zurücksetzen.



Anmerkung

Wenn mit dem Parameter `--to-version` keine Revision angegeben ist, wird die letzte erfolgreich bereitgestellte Revision verwendet.



Anmerkung

Deploymentkonfigurationen unterstützen das automatische Rollback zur letzten erfolgreichen Revision der Konfiguration, falls der letzte Deploymentprozess fehlschlägt. In diesem Fall bleibt die letzte Vorlage, deren Bereitstellung fehlgeschlagen ist, unverändert und kann überprüft werden.

Dieses Feature ist derzeit nicht für Deployment-Ressourcen verfügbar, wo von Ihnen erwartet wird, dass Builds durch Festlegen von Tags erneut bereitgestellt werden. Wenn Sie mit Deployment-Ressourcen arbeiten, können Sie Hashes von Builds finden, indem Sie `oc describe imagestream name` verwenden. Legen Sie anschließend mit dem Befehl `oc tag imagename:latest hash-from-older-build` den Hash als Tag fest.

- Um zu verhindern, dass ein neuer Deploymentprozess versehentlich gestartet wird, nachdem ein Rollback abgeschlossen wurde, werden die Trigger für Image-Änderungen im Rahmen des Rollback-Prozesses deaktiviert.

Sie können diese Trigger jedoch nach dem Rollback mit dem Befehl `oc set triggers` wieder aktivieren:

```
[user@host ~]$ oc set triggers dc/name --auto
```

- Führen Sie zum Anzeigen der DeploymentLogs den Befehl `oc logs` aus:

```
[user@host ~]$ oc logs -f dc/name
```

Wenn die neueste Revision ausgeführt wird oder fehlgeschlagen ist, gibt der Befehl `oc logs` die Logs des Prozesses zurück, der für das Bereitstellen Ihrer Pods verantwortlich ist. Wenn sie erfolgreich ist, werden die Logs eines Pods Ihrer Anwendung zurückgegeben.

Sie können auch die Logs von älteren fehlgeschlagenen Deploymentprozessen anzeigen, sofern sie weder gekürzt noch manuell gelöscht wurden:

```
[user@host ~]$ oc logs --version=1 dc/name
```

- Sie können die Anzahl der Pods in einer Bereitstellung durch Ausführen des Befehls `oc scale` skalieren:

```
[user@host ~]$ oc scale dc/name --replicas=3
```

Die Anzahl der Replikate wird schließlich an den gewünschten und aktuellen Zustand weitergeleitet, der von der Deploymentkonfiguration konfiguriert wurde.

Trigger für Bereitstellungen

Eine Deploymentkonfiguration kann Trigger enthalten, die als Reaktion auf Ereignisse in oder außerhalb von Red Hat OpenShift die Neuerstellung von Bereitstellungen auslösen. Es gibt zwei Arten von Ereignissen, die eine Bereitstellung auslösen:

- Konfigurationsänderung
- Image-Änderung

Trigger für Konfigurationsänderungen

Der Trigger `ConfigChange` führt immer dann zu einer neuen Bereitstellung, wenn Änderungen an der Replication Controller-Vorlage der Deploymentkonfiguration ermittelt werden. Sie können sich darauf verlassen, dass dieser Trigger aktiviert wird, nachdem Sie die Replikatgröße geändert, das Image für die Anwendung geändert oder andere Änderungen an der Deploymentkonfiguration durchgeführt haben.

Ein Beispiel eines `ConfigChange`-Triggers wird im Folgenden gezeigt:

```
triggers:  
  - type: "ConfigChange"
```

Trigger für Image-Änderungen

Der Trigger `ImageChange` führt immer dann zu einer neuen Bereitstellung, wenn sich der Wert eines Image-Stream-Tags ändert. Dies ist in einer Umgebung nützlich, in der die Images sicherheitshalber von der Anwendung oder von Library-Aktualisierungen unabhängig aktualisiert werden.

Ein Beispiel eines ImageChange-Triggers wird im Folgenden gezeigt:

```
triggers:
  - type: "ImageChange"
    imageChangeParams:
      automatic: true①
      containerNames:
        - "helloworld"
    from:
      kind: "ImageStreamTag"
      name: "origin-ruby-sample:latest"
```

- ① Wenn das Attribut `automatic` auf "false" festgelegt ist, wird der Trigger deaktiviert.

Im vorherigen Beispiel wird, wenn der Tag-Wert `latest` des Image-Streams `origin-ruby-sample` geändert wird, eine neue Bereitstellung mit dem neuen Tag-Wert für „container“ erstellt.

Führen Sie den Befehl `oc set triggers dc/name \--from-image=myproject/origin-ruby-sample:latest -c helloworld`

Festlegen von Deployment-Ressourcen-Limits

Eine Bereitstellung wird durch einen Pod abgeschlossen, der Ressourcen (Arbeitsspeicher und CPU) auf einem Knoten in Anspruch nimmt. Pods nehmen standardmäßig unbegrenzte Knotenressourcen in Anspruch. Wenn ein Projekt jedoch standardmäßige Ressourcengrenzen angibt, nehmen Pods nur die Ressourcen bis zu diesen Grenzen in Anspruch.

Sie können auch die Ressourcennutzung begrenzen, indem Sie im Rahmen der Deploymentstrategie Ressourcengrenzen angeben. Diese Ressourcengrenzen gelten für die durch die Bereitstellung erstellten Anwendungs-Pods, jedoch nicht für Deployer-Pods. Sie können Deployment-Ressourcen zusammen mit Recreate-, Rolling- oder benutzerdefinierten Deploymentstrategien verwenden.

Im folgenden Beispiel werden die für die Bereitstellung erforderlichen Ressourcen unter dem Attribut `resources` der Deploymentkonfiguration deklariert:

```
type: "Recreate"
resources:
  limits:
    cpu: "100m" ①
    memory: "256Mi" ②
```

- ① CPU-Ressource in CPU-Einheiten. `100m` entsprechen 0,1 CPU-Einheiten.
② Arbeitsspeicherressourcen in Byte. `256Mi` entspricht 268435456 Byte ($256 * 2^20$).



Literaturhinweise

Zusätzliche Informationen zu Bereitstellungen finden Sie im Kapitel *Deployments* im Handbuch *Applications* für Red Hat OpenShift Container Platform 4.6 unter
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/applications/index#deployments

► Angeleitete Übung

Verwalten der Anwendungsbereitstellungen

In dieser Übung verwalten Sie die Bereitstellung einer Anwendung, die in einem Red Hat OpenShift-Cluster ausgeführt wird.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Bereitstellen einer Thorntail-basierten Anwendung auf einem Red Hat OpenShift-Cluster
- Aktualisieren der Deploymentkonfiguration für die ausgeführte Anwendung zum Einbeziehen eines Liveness-Probes
- Nehmen Sie Änderungen am Anwendungsquellcode vor und stellen Sie die Anwendung erneut bereit.
- Zurücksetzen der Anwendung auf die zuvor bereitgestellte Version

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen ausgeführten Red Hat OpenShift-Cluster
- Auf das OpenJDK-S2I-Builder-Image `redhat-openjdk-18/openjdk18-openshift`
- Auf die Anwendung `quip` im Git-Repository

Führen Sie den folgenden Befehl auf `workstation` aus, um die Voraussetzungen für die Übung zu überprüfen und die Lab- und Lösungsdateien herunterzuladen:

```
[student@workstation ~]$ lab app-deploy start
```

Anweisungen

► 1. Überprüfen Sie den Anwendungsquellcode.

- 1.1. Wechseln Sie zu Ihrem lokalen Klon des Git-Repositories `D0288-apps`, und checken Sie den `master`-Branch des Kurs-Repositorys aus, um sicherzustellen, dass Sie diese Übung von einem als fehlerfrei bekannten Zustand aus starten:

```
[student@workstation ~]$ cd D0288-apps
[student@workstation D0288-apps]$ git checkout main
...output omitted...
```

- 1.2. Erstellen Sie einen neuen Branch, in dem Sie alle Änderungen speichern können, die Sie während dieser Übung vornehmen:

```
[student@workstation D0288-apps]$ git checkout -b app-deploy
Switched to a new branch 'app-deploy'
[student@workstation D0288-apps]$ git push -u origin app-deploy
...output omitted...
* [new branch]      app-deploy -> app-deploy
Branch app-deploy set up to track remote branch app-deploy from origin.
[student@workstation D0288-apps]$ cd
```

13. Überprüfen Sie die Datei ~/D0288-apps/quip/src/main/java/com/redhat/training/example/Quip.java.

Die quip-Anwendung ist eine REST-Serviceimplementierung für JAX-RS von Java mit zwei Endpunkten:

```
...output omitted...
@Path("/")
public class Quip {

    @GET
    @Produces("text/plain")
    public Response index() throws Exception {
        String host = InetAddress.getLocalHost().getHostName();
        return Response.ok("Veni, vidi, vici...\n").build();①
    }

    @GET
    @Path("/ready")
    @Produces("text/plain")
    public Response ready() throws Exception {
        return Response.ok("OK\n").build();②
    }
...output omitted...
```

- ① Bei an den Endpunkt / gesendeten Anforderungen wird ein Zitat zurückgegeben
- ② Bei an den Endpunkt /ready gesendeten Anforderungen wird die Meldung OK zurückgegeben.

► 2. Erstellen Sie eine auf dem Quellcode basierende Anwendung.

- 2.1. Führen Sie mit dem Befehl „source“ die Konfiguration Ihrer Kursumgebung aus:

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

- 2.2. Melden Sie sich bei Red Hat OpenShift mit Ihrem Entwicklerbenutzerkonto an:

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful
...output omitted...
```

- 2.3. Erstellen Sie ein neues Projekt für die Anwendung. Stellen Sie dem Projektnamen Ihren Entwicklerbenutzernamen voran:

```
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-app-deploy
```

2.4. Erstellen Sie eine Anwendung mit den folgenden Parametern:

- Name: quip
- Build-Umgebungsvariablen:
 - Name: MAVEN_MIRROR_URL, Wert: http://\${RHT_OCP4_NEXUS_SERVER}/repository/java
- Image-Stream: redhat-openjdk18-openshift:1.5
- Verzeichnis der Anwendung: /quip

Sie können das Skript /home/student/D0288/labs/app-deploy/oc-new-app.sh oder den folgenden Befehl ausführen:

```
[student@workstation ~]$ oc new-app --as-deployment-config --name quip \
--build-env MAVEN_MIRROR_URL=http://${RHT_OCP4_NEXUS_SERVER}/repository/java \
-i redhat-openjdk18-openshift:1.5 --context-dir quip \
https://github.com/${RHT_OCP4_GITHUB_USER}/D0288-apps#app-deploy
--> Found image c1bf724 (9 months old) in image stream "openshift/redhat-
openjdk18-...
--> Creating resources ...
imagestream.image.openshift.io "quip" created
buildconfig.build.openshift.io "quip" created
deploymentconfig.apps.openshift.io "quip" created
service "quip" created
--> Success
...output omitted...
```

2.5. Zeigen Sie die Anwendungs-Build-Logs an. Das Erstellen des Anwendungscontainer-Images und das Übertragen an die interne Red Hat OpenShift-Registry dauert eine Weile:

```
[student@workstation ~]$ oc logs -f bc/quip
...output omitted...
[INFO] Repackaged .war: /tmp/src/target/quip.war
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
...output omitted...
Pushing image ....registry.svc:5000/youruser-app-deploy/quip:latest ...
...output omitted...
Push successful
```

2.6. Warten Sie, bis die Anwendung bereitgestellt wird. Der Anwendungs-Pod muss den Status READY aufweisen:

```
[student@workstation ~]$ oc get pods
NAME        READY   STATUS    RESTARTS   AGE
quip-1-59j8q   1/1     Running   0          10s
quip-1-build   0/1     Completed  0          89s
quip-1-deploy  0/1     Completed  0          12s
```

- 3. Testen Sie die Anwendung, um zu verifizieren, dass sie Anforderungen von Clients verarbeitet.
- 3.1. Überprüfen Sie die AnwendungsLogs, um zu erfahren, ob während des Starts Fehler auftreten:

```
[student@workstation ~]$ oc get pods
NAME        READY   STATUS    RESTARTS   AGE
quip-1-59j8q   1/1     Running   0          50s
...output omitted...
[student@workstation ~]$ oc logs quip-1-59j8q
...output omitted...
... INFO [org.jboss.as.server] (main) WFLYSRV0010: Deployed "quip.war" (...)
... INFO [org.wildfly.swarm] (main) WFSWARM99999: Thorntail is Ready
```

Die Logs zeigen, dass die Anwendung ohne Fehler gestartet wurde.

- 3.2. Verifizieren Sie, dass die Service-Ressource für die Anwendung einen registrierten Endpunkt zum Weiterleiten eingehender Anforderungen aufweist:

```
[student@workstation ~]$ oc describe svc/quip
Name:           quip
...output omitted...
IP:             172.30.37.127
Port:           8080-tcp  8080/TCP
TargetPort:     8080/TCP
Endpoints:     10.128.2.111:8080
...output omitted...
```

- 3.3. Stellen Sie die Anwendung für den externen Zugriff mit einer Route bereit:

```
[student@workstation ~]$ oc expose svc quip
route.route.openshift.io/quip exposed
```

- 3.4. Testen Sie die Anwendung mit der Routen-URL, die Sie im vorherigen Schritt abgerufen haben:

```
[student@workstation ~]$ curl \
http://quip-${RHT_OCP4_DEV_USER}-app-deploy.${RHT_OCP4_WILDCARD_DOMAIN}
Veni, vidi, vici...
```

- 4. Aktivieren Sie die Readiness- und Liveness-Probes für die Anwendung.
- 4.1. Führen Sie den Befehl `oc set` aus, um einen Liveness- und Readiness-Probe mit den folgenden Parametern für beide Tests zur DeploymentConfig-Ressource hinzuzufügen.

- Endpunkt: /ready
- Port: 8080
- Anfangsverzögerung: 30 Sekunden
- Timeout: 2 Sekunden

```
[student@workstation ~]$ oc set probe dc/quip \
--liveness --readiness --get-url=http://:8080/ready \
--initial-delay-seconds=30 --timeout-seconds=2
deploymentconfig.apps.openshift.io/quip probes updated
```

- 4.2. Verifizieren Sie den Wert in den Einträgen livenessProbe und readinessProbe:

```
[student@workstation ~]$ oc describe dc/quip | grep http-get
Liveness:    http-get http://:8080/ready delay=30s timeout=2s period=10s
#success=1 #failure=3
Readiness:   http-get http://:8080/ready delay=30s timeout=2s period=10s
#success=1 #failure=3
```

- 4.3. Warten Sie, bis der Anwendungs-Pod erneut bereitgestellt wird und im Status READY angezeigt wird:

```
[student@workstation ~]$ oc get pods
...output omitted...
quip-2-n6nzw     1/1      Running     0          26s
```

- 4.4. Führen Sie den Befehl `oc describe` aus, um den ausgeführten Pod zu überprüfen und um sicherzustellen, dass die Probes aktiv sind:

```
[student@workstation ~]$ oc describe pod quip-2-n6nzw | grep http-get
Liveness:    http-get http://:8080/ready delay=30s timeout=2s...
Readiness:   http-get http://:8080/ready delay=30s timeout=2s...
```

Die Readiness- und Liveness-Probes für die Anwendung sind nun aktiv.

- 4.5. Testen Sie die Anwendung mit der Routen-URL, die Sie im vorherigen Schritt abgerufen haben:

```
[student@workstation ~]$ curl \
http://quip-${RHT_OCP4_DEV_USER}-app-deploy.${RHT_OCP4_WILDCARD_DOMAIN}
Veni, vidi, vici...
```

- 5. Nehmen Sie Änderungen am Anwendungsquellcode vor und stellen Sie die Anwendung erneut bereit.

Verifizieren Sie, dass Sie beim Testen der Anwendung die Änderungen sehen können.

- 5.1. Überprüfen Sie das Skript `~/D0288/labs/app-deploy/app-change.sh`.

Das Skript ändert den Quellcode, damit die Meldung in Englisch ausgegeben wird, committet und überträgt dann die Änderung an das Git-Repository.

Kapitel 7 | Verwalten der Anwendungsbereitstellungen

```
[student@workstation ~$ cat ~/D0288/labs/app-deploy/app-change.sh
#!/bin/bash

echo "Changing quip to english..."
sed -i 's/Veni, vidi, vici/I came, I saw, I conquered/g' \
/home/student/D0288-apps/quip/src/main/java/com/redhat/training/example/Quip.java

echo "Committing the changes..."
cd /home/student/D0288-apps/quip
git commit -a -m "Changed quip lang to english"

echo "Pushing changes to classroom Git repository..."
git push
cd
```

5.2. Führen Sie das Skript `~/D0288/labs/app-deploy/app-change.sh` aus:

```
[student@workstation ~]$ ~/D0288/labs/app-deploy/app-change.sh
Changing quip to english...
Committing the changes...
[app-deploy afd7c3] Changed quip lang to english
...output omitted...
To https://github.com/youruser/D0288-apps
  dfe07f7..0aa1ac1  app-deploy -> app-deploy
```

5.3. Starten Sie einen neuen Build der Anwendung, und beobachten Sie die Build-Logs:

```
[student@workstation ~]$ oc start-build quip -F
build.build.openshift.io/quip-2 started
...output omitted...
Push successful
```

5.4. Warten Sie, bis ein neuer Anwendungs-Pod bereitgestellt wird. Der Pod muss den Status READY aufweisen:

NAME	READY	STATUS	RESTARTS	AGE
quip-1-build	0/1	Completed	0	12m
quip-1-deploy	0/1	Completed	0	11m
quip-2-build	0/1	Completed	0	2m11s
quip-2-deploy	0/1	Completed	0	4m59s
quip-3-deploy	0/1	Completed	0	60s
quip-3-gvzs5	1/1	Running	0	56s

5.5. Testen Sie die Anwendung nach der Änderung erneut, und verifizieren Sie, dass eine Meldung in Englisch ausgegeben wird:

```
[student@workstation ~]$ curl \
http://quip-${RHT_OCP4_DEV_USER}-app-deploy.${RHT_OCP4_WILDCARD_DOMAIN}
I came, I saw, I conquered...
```

- 6. Nehmen Sie ein Rollback auf die vorherige Bereitstellung vor.

Überprüfen Sie, ob die vorherige Meldung `Veni, vidi, vici...` angezeigt wird.

- 6.1. Nehmen Sie ein Rollback auf die vorherige Version der Bereitstellung vor.

Es wird eine Warnung angezeigt, dass die Trigger für Image-Änderungen durch den Befehl `oc rollback` deaktiviert wurden.

```
[student@workstation ~]$ oc rollback dc/quip
deploymentconfig.apps.openshift.io/quip deployment #4 rolled back to quip-2
Warning: the following images triggers were disabled: quip:latest
You can re-enable them with: oc set triggers dc/quip --auto
```

- 6.2. Warten Sie, bis der neue Anwendungs-Pod bereitgestellt wird. Er muss den Status `READY` aufweisen:

```
[student@workstation ~]$ oc get pods
NAME        READY   STATUS    RESTARTS   AGE
quip-1-build  0/1     Completed  0          17m
quip-1-deploy 0/1     Completed  0          16m
quip-2-build  0/1     Completed  0          7m1s
quip-2-deploy 0/1     Completed  0          9m49s
quip-3-deploy 0/1     Completed  0          5m50s
quip-4-9h6ql  1/1     Running   0          89s
quip-4-deploy 0/1     Completed  0          95s
```

- 6.3. Nachdem Sie die Anwendung zurückgesetzt haben, testen Sie sie erneut, und verifizieren Sie, dass die lateinische Meldung ausgegeben wird:

```
[student@workstation ~]$ curl \
http://quip-${RHT_OCP4_DEV_USER}-app-deploy.${RHT_OCP4_WILDCARD_DOMAIN}
Veni, vidi, vici...
```

- 7. Führen Sie eine Bereinigung durch. Löschen Sie das Projekt.

```
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-app-deploy
project.project.openshift.io "youruser-app-deploy" deleted
```

Beenden

Führen Sie auf der `workstation` den Befehl `lab app-deploy finish` aus, um diese Übung zu beenden. Dieser Schritt ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken.

```
[student@workstation ~]$ lab app-deploy finish
```

Hiermit ist die angeleitete Übung beendet.

► Praktische Übung

Verwalten der Anwendungsbereitstellungen

In diesem Lab verwalten Sie die Bereitstellung einer Anwendung und skalieren sie auf einem Red Hat OpenShift Container Platform-Cluster (RHOCP).

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Bereitstellen einer PHP-basierten Anwendung auf einem Red Hat OpenShift-Cluster
- Skalieren der Anwendung für die Ausführung in mehreren Pods
- Ändern der Anwendungsquelle, erneutes Bereitstellen der Anwendung und Verifizieren, dass die Änderungen berücksichtigt wurden
- Zurücksetzen der Änderung und Verifizieren, dass die vorherige Version der Anwendung bereitgestellt wird

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um dieses Lab durchführen zu können:

- Auf einen ausgeführten Red Hat OpenShift-Cluster
- Auf den `php-scale`-Anwendungsquellcode im Git-Repository-Fork D0288-apps

Führen Sie den folgenden Befehl auf `workstation` aus, um die Voraussetzungen zu überprüfen:

```
[student@workstation ~]$ lab manage-deploy start
```

Anforderungen

In dieser praktischen Übung wird eine PHP-basierte Anwendung verwendet, die folgende Informationen ausgibt:

- Die Version der Anwendung
- Den Namen des Anwendungs-Pods
- Die IP-Adresse des Anwendungs-Pods

Stellen Sie die Anwendung entsprechend den folgenden Anweisungen in einem Red Hat OpenShift-Cluster bereit, und testen Sie sie:

- Verwenden Sie den Image-Stream `php:7.3`, um die Anwendung bereitzustellen.
- Stellen Sie sicher, dass der Anwendungsname für Red Hat OpenShift `scale` lautet.
- Erstellen Sie die Anwendung in einem Projekt mit dem Namen `youruser-manage-deploy`.

- Stellen Sie sicher, dass der Zugriff auf die Anwendung möglich ist über die URL:
`http://scale-youruser-manage-deploy.apps.cluster.domain.example.com` .
- Stellen Sie sicher, dass die Anwendung das Git-Repository verwendet unter:
`https://github.com/youruser/D0288-apps`.
- Das Verzeichnis `php-scale` im Git-Repository enthält den Quellcode für die Anwendung.
- Stellen Sie sicher, dass die Anwendung die `DeploymentConfig`-Ressource verwendet.

Anweisungen

1. Wechseln Sie zu Ihrem lokalen Klon des Git-Repositorys `D0288-apps`, und checken Sie den `master`-Branch des Kurs-Repositorys aus, um sicherzustellen, dass Sie diese Übung von einem als fehlerfrei bekannten Zustand aus starten:

```
[student@workstation ~]$ cd D0288-apps
[student@workstation D0288-apps]$ git checkout main
...output omitted...
```

2. Erstellen Sie einen neuen Branch, in dem Sie alle Änderungen speichern können, die Sie während dieser Übung vornehmen:

```
[student@workstation D0288-apps]$ git checkout -b manage-deploy
Switched to a new branch 'manage-deploy'
[student@workstation D0288-apps]$ git push -u origin manage-deploy
...output omitted...
* [new branch]      manage-deploy -> manage-deploy
Branch manage-deploy set up to track remote branch manage-deploy from origin.
```

3. Melden Sie sich mit Ihrem persönlichen Entwicklerbenutzernamen beim Red Hat OpenShift-Cluster an.

Erstellen Sie ein neues Projekt mit dem Namen `youruser-manage-deploy`.

Stellen Sie die Anwendung mithilfe des Image-Streams `php:7.3` im Verzeichnis `php-scale` bereit. Nennen Sie die Anwendung `scale`, und verwenden Sie die `DeploymentConfig`-Ressource zum Verwalten der Anwendung.

Stellen Sie beim Bereitstellen der Anwendung sicher, dass Sie auf den Branch `manage-deploy` verweisen, den Sie im vorherigen Schritt erstellt haben.

Stellen Sie die Anwendung mit der generierten Routen-URL bereit, und testen Sie sie. Verifizieren Sie, dass Sie `version 1` und den Pod-Namen in der Ausgabe sehen können.

4. Verifizieren Sie, dass die `Rolling`-Strategie die StandardDeploymentstrategie ist.

5. Skalieren Sie die Anwendung auf zwei Pods.

Führen Sie den Befehl `curl` aus, um die Anwendung erneut zu testen, und verifizieren Sie, dass für die Anforderungen in den zwei Pods eine Round-Robin-Lastverteilung besteht.

6. Ändern Sie die Versionsnummer in der Datei `index.php` in 2. Bestätigen und übertragen Sie die Änderungen an das Remote-Git-Repository.

Nehmen Sie keine weiteren Änderungen am Quellcode vor.

Kapitel 7 | Verwalten der Anwendungsbereitstellungen

- Starten Sie einen neuen Build der Anwendung.

Verifizieren Sie, dass Red Hat OpenShift die Pods, auf denen die ältere Version ausgeführt wird, herunterskaliert, und dass es die neuen Pods mit der neuesten Version der Anwendung hochskaliert.

Testen Sie die Anwendung erneut mit dem Befehl `curl`. Verifizieren Sie, dass `version 2` in der Ausgabe angezeigt wird.

- Nehmen Sie ein Rollback auf die vorherige Bereitstellung vor.

Führen Sie den Befehl `curl` aus, um die Anwendung erneut zu testen. Verifizieren Sie, dass `version 1` in der Ausgabe angezeigt wird.

- Bewerten Sie Ihre Arbeit:

```
[student@workstation D0288-apps]$ lab manage-deploy grade
```

- Bereinigen und löschen Sie das Projekt.

```
[student@workstation D0288-apps]$ oc delete project \
${RHT_OCP4_DEV_USER}-manage-deploy
```

Beenden

Führen Sie auf `workstation` das Skript `lab manage-deploy finish` aus, um diese Übung zu beenden. Dieser Schritt ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken. Die Aktion „Beenden“ veröffentlicht dieses Projekt und seine Ressourcen.

```
[student@workstation ~]$ lab manage-deploy finish
```

Dadurch wird die Wiederholungsübung abgeschlossen.

► Lösung

Verwalten der Anwendungsbereitstellungen

In diesem Lab verwalten Sie die Bereitstellung einer Anwendung und skalieren sie auf einem Red Hat OpenShift Container Platform-Cluster (RHOCP).

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Bereitstellen einer PHP-basierten Anwendung auf einem Red Hat OpenShift-Cluster
- Skalieren der Anwendung für die Ausführung in mehreren Pods
- Ändern der Anwendungsquelle, erneutes Bereitstellen der Anwendung und Verifizieren, dass die Änderungen berücksichtigt wurden
- Zurücksetzen der Änderung und Verifizieren, dass die vorherige Version der Anwendung bereitgestellt wird

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um dieses Lab durchführen zu können:

- Auf einen ausgeführten Red Hat OpenShift-Cluster
- Auf den `php-scale`-Anwendungsquellcode im Git-Repository-Fork D0288-apps

Führen Sie den folgenden Befehl auf `workstation` aus, um die Voraussetzungen zu überprüfen:

```
[student@workstation ~]$ lab manage-deploy start
```

Anforderungen

In dieser praktischen Übung wird eine PHP-basierte Anwendung verwendet, die folgende Informationen ausgibt:

- Die Version der Anwendung
- Den Namen des Anwendungs-Pods
- Die IP-Adresse des Anwendungs-Pods

Stellen Sie die Anwendung entsprechend den folgenden Anweisungen in einem Red Hat OpenShift-Cluster bereit, und testen Sie sie:

- Verwenden Sie den Image-Stream `php:7.3`, um die Anwendung bereitzustellen.
- Stellen Sie sicher, dass der Anwendungsname für Red Hat OpenShift `scale` lautet.
- Erstellen Sie die Anwendung in einem Projekt mit dem Namen `youruser-manage-deploy`.

Kapitel 7 | Verwalten der Anwendungsbereitstellungen

- Stellen Sie sicher, dass der Zugriff auf die Anwendung möglich ist über die URL:
`http://scale-youruser-manage-deploy.apps.cluster.domain.example.com`.
- Stellen Sie sicher, dass die Anwendung das Git-Repository verwendet unter:
`https://github.com/youruser/D0288-apps`.
- Das Verzeichnis `php-scale` im Git-Repository enthält den Quellcode für die Anwendung.
- Stellen Sie sicher, dass die Anwendung die `DeploymentConfig`-Ressource verwendet.

Anweisungen

1. Wechseln Sie zu Ihrem lokalen Klon des Git-Repositories `D0288-apps`, und checken Sie den `master`-Branch des Kurs-Repositorys aus, um sicherzustellen, dass Sie diese Übung von einem als fehlerfrei bekannten Zustand aus starten:

```
[student@workstation ~]$ cd D0288-apps  
[student@workstation D0288-apps]$ git checkout main  
...output omitted...
```

2. Erstellen Sie einen neuen Branch, in dem Sie alle Änderungen speichern können, die Sie während dieser Übung vornehmen:

```
[student@workstation D0288-apps]$ git checkout -b manage-deploy  
Switched to a new branch 'manage-deploy'  
[student@workstation D0288-apps]$ git push -u origin manage-deploy  
...output omitted...  
 * [new branch]      manage-deploy -> manage-deploy  
Branch manage-deploy set up to track remote branch manage-deploy from origin.
```

3. Melden Sie sich mit Ihrem persönlichen Entwicklerbenutzernamen beim Red Hat OpenShift-Cluster an.

Erstellen Sie ein neues Projekt mit dem Namen `youruser-manage-deploy`.

Stellen Sie die Anwendung mithilfe des Image-Streams `php:7.3` im Verzeichnis `php-scale` bereit. Nennen Sie die Anwendung `scale`, und verwenden Sie die `DeploymentConfig`-Ressource zum Verwalten der Anwendung.

Stellen Sie beim Bereitstellen der Anwendung sicher, dass Sie auf den Branch `manage-deploy` verweisen, den Sie im vorherigen Schritt erstellt haben.

Stellen Sie die Anwendung mit der generierten Routen-URL bereit, und testen Sie sie. Verifizieren Sie, dass Sie `version 1` und den Pod-Namen in der Ausgabe sehen können.

3.1. Führen Sie mit dem Befehl „source“ die Konfiguration Ihrer Kursumgebung aus:

```
[student@workstation D0288-apps]$ source /usr/local/etc/ocp4.config
```

3.2. Melden Sie sich bei Red Hat OpenShift an, und erstellen Sie ein neues Projekt. Stellen Sie dem Projektnamen Ihren Entwicklerbenutzernamen voran.

```
[student@workstation D0288-apps]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
[student@workstation D0288-apps]$ oc new-project \
${RHT_OCP4_DEV_USER}-manage-deploy
Now using project "yourusername-manage-deploy" on server "https://
api.cluster.domain.example.com:6443".
...output omitted...
```

- 3.3. Erstellen Sie eine neue Anwendung:

```
[student@workstation D0288-apps]$ oc new-app --as-deployment-config --name scale \
php:7.3~https://github.com/${RHT_OCP4_GITHUB_USER}/D0288-apps#manage-deploy \
--context-dir php-scale
--> Found image f10275b (3 weeks old) in image stream "openshift/php" under...
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "scale" created
buildconfig.build.openshift.io "scale" created
deploymentconfig.apps.openshift.io "scale" created
service "scale" created
--> Success
...output omitted...
```

- 3.4. Sehen Sie sich die Build-Logs an. Warten Sie, bis der Build abgeschlossen ist und das Anwendungscontainer-Image an die Red Hat OpenShift-Registry übertragen wurde:

```
[student@workstation D0288-apps]$ oc logs -f bc/scale
...output omitted...
Push successful
```

- 3.5. Warten Sie, bis die Anwendung bereitgestellt wird. Der Anwendungs-Pod sollte den Status READY aufweisen:

```
[student@workstation D0288-apps]$ oc get pods
NAME        READY   STATUS    RESTARTS   AGE
scale-1-build  0/1     Completed   0          5m14s
scale-1-deploy  0/1     Completed   0          82s
scale-1-w48nd   1/1     Running    0          74s
```

- 3.6. Verwenden Sie eine Route, um die Anwendung für den externen Zugriff bereitzustellen:

```
[student@workstation D0288-apps]$ oc expose svc scale
route.route.openshift.io/scale exposed
```

- 3.7. Testen Sie die Anwendung mit dem Befehl curl:

```
[student@workstation D0288-apps]$ curl \
http://scale-${RHT_OCP4_DEV_USER}-manage-deploy.${RHT_OCP4_WILDCARD_DOMAIN}
This is version 1 of the app. I am running on host -> scale-1-w48nd (10.128.1.21)
```

4. Verifizieren Sie, dass die Rolling-Strategie die StandardDeploymentstrategie ist.

```
[student@workstation D0288-apps]$ oc describe dc/scale | grep -i strategy
Strategy: Rolling
```

5. Skalieren Sie die Anwendung auf zwei Pods.

Führen Sie den Befehl `curl` aus, um die Anwendung erneut zu testen, und verifizieren Sie, dass für die Anforderungen in den zwei Pods eine Round-Robin-Lastverteilung besteht.

- 5.1. Öffnen Sie die Red Hat OpenShift-Webkonsole-URL in einem Webbrowser:

```
[student@workstation D0288-apps]$ oc get route console -n openshift-console \
-o jsonpath='{.spec.host}{"\n"}'
console-openshift-console.apps.cluster.domain.example.com
```

- 5.2. Melden Sie sich als Benutzer developer an.

Überprüfen Sie Ihre Anmelddaten mithilfe der Datei `/usr/local/etc/ocp4.config`:

- Der Benutzername ist der Wert der Variablen `RHT_OCP4_DEV_USER`.
- Das Passwort ist der Wert der Variablen `RHT_OCP4_DEV_PASSWORD`.

- 5.3. Wenn Sie sich nicht in der Administratoransicht befinden, klicken Sie auf **Developer** → **Administrator**, um zur Administratoransicht zu wechseln.

Click `youruser-manage-deploy` on the **Projects** page to open the **Overview** page for the project.

Klicken Sie auf den Tab **Workloads**, um die für das Projekt verfügbaren Bereitstellungen anzuzeigen.

- 5.4. Wählen Sie den Eintrag **scale** der Deploymentkonfiguration aus.

Klicken Sie auf den Abschnitt **Details**. Klicken Sie dann auf den oberen Pfeil auf der rechten Seite des blauen Kreises, um die Anzahl der Pods auf zwei zu erhöhen.

The screenshot shows the Red Hat OpenShift Workloads interface. At the top, there's a navigation bar with 'Projects' (highlighted in blue), 'Project Details', and a search bar. Below that is a header for a project named 'PR youruser-manage-deploy' with an 'Active' status and an 'Actions' dropdown. The main area has tabs for 'Overview', 'Details', 'YAML', 'Workloads' (which is selected and highlighted in blue), and 'Role Bindings'. Under 'Workloads', there's a section for 'scale' with a sub-section for 'DC scale, #1'. It shows '1 of 1 pods'. A large blue circle contains the number '1 pod'. To the right of the circle is a small red square with a white upward-pointing arrow. Below the circle, there are columns for 'Name' (scale) and 'Latest Version' (1). On the left side, there's a sidebar with a 'Group by: Application' dropdown and a 'Filter by name...' input field. A note at the bottom says: '1 and 2 selects items, and 3 filters items.'

Beobachten Sie, wie Red Hat OpenShift einen weiteren Pod erstellt. Dies kann einige Minuten dauern.

- 5.5. Kehren Sie zum Terminalfenster zurück, und führen Sie den Befehl `curl` aus, um mehrere HTTP-Anforderungen zur Routen-URL zu senden, um die Anwendung zu testen:

```
[student@workstation D0288-apps]$ curl \
http://scale-${RHT_OCP4_DEV_USER}-manage-deploy.${RHT_OCP4_WILDCARD_DOMAIN}
This is version 1 of the app. I am running on host -> scale-1-gp3w0 (10.128.1.21)
[student@workstation D0288-apps]$ curl \
http://scale-${RHT_OCP4_DEV_USER}-manage-deploy.${RHT_OCP4_WILDCARD_DOMAIN}
This is version 1 of the app. I am running on host -> scale-1-567x7 (10.129.1.101)
```

Sie sollten sehen, dass für die Anforderungen in den zwei Pods eine Round-Robin-Lastverteilung besteht.



Anmerkung

Sie können die Routen-URL nicht mit einem Webbrowser testen, da der Red Hat OpenShift-Router standardmäßig die Sitzungsaffinität aktiviert. Daher können Sie das Lastenausgleichsverhalten nicht verifizieren. Testen Sie die Anwendung mit dem Befehl `curl`:

6. Ändern Sie die Versionsnummer in der Datei `index.php` in 2. Bestätigen und übertragen Sie die Änderungen an das Remote-Git-Repository.

Nehmen Sie keine weiteren Änderungen am Quellcode vor.

Kapitel 7 | Verwalten der Anwendungsbereitstellungen

- 6.1. Bearbeiten Sie die Datei `~/D0288-apps/php-scale/index.php`, und ändern Sie die Versionsnummer (in der zweiten Zeile) in 2, wie dies im Folgenden gezeigt wird. Ändern Sie nichts anderes in dieser Datei:

```
<?php  
print "This is version 2 of the app. I am running on host...  
?>
```

- 6.2. Bestätigen und übertragen Sie die Änderungen an das Git-Repository:

```
[student@workstation D0288-apps]$ git commit -a -m "Updated app to version 2"  
[manage-deploy 3633f74] updating version  
 1 file changed, 1 insertion(+), 1 deletion(-)  
[student@workstation D0288-apps]$ git push  
...output omitted...
```

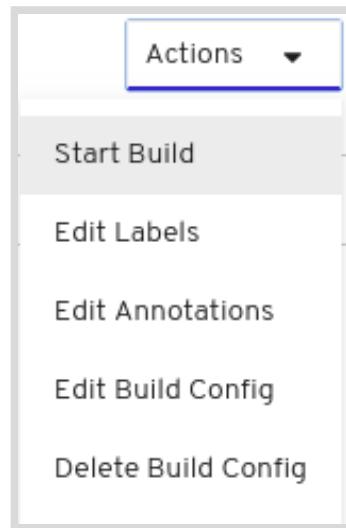
7. Starten Sie einen neuen Build der Anwendung.

Verifizieren Sie, dass Red Hat OpenShift die Pods, auf denen die ältere Version ausgeführt wird, herunterskaliert, und dass es die neuen Pods mit der neuesten Version der Anwendung hochskaliert.

Testen Sie die Anwendung erneut mit dem Befehl `curl`. Verifizieren Sie, dass `version 2` in der Ausgabe angezeigt wird.

- 7.1. Klicken Sie im linken Menü des Red Hat OpenShift-Webkonsole auf den Eintrag **Builds** → **Build Configs**.

Wählen Sie die Build-Konfiguration **scale** aus, und klicken Sie auf **Actions** → **Start Build**.



- 7.2. Die Konsole leitet Sie zur Zusammenfassungsseite über den neuen Build um. Klicken Sie auf den Tab **Logs**, um sich das Build-Log anzusehen.
- 7.3. Nach Abschluss des Builds skaliert Red Hat OpenShift zwei neue Pods der neuen Version der Anwendung hoch.

Wenn die neuen Pods Datenverkehr empfangen können, skaliert Red Hat OpenShift die beiden älteren Pods herunter.

Klicken Sie auf **Workloads** → **Pods**, um die neuen Anwendungs-Pods anzuzeigen.

- 7.4. Kehren Sie zum Terminalfenster zurück, und führen Sie den Befehl `curl` aus, um mehrere HTTP-Anforderungen zur Routen-URL zu senden, um die Anwendung zu testen:

```
[student@workstation D0288-apps]$ curl \
http://scale-${RHT_OCP4_DEV_USER}-manage-deploy.${RHT_OCP4_WILDCARD_DOMAIN}
This is version 2 of the app. I am running on host -> scale-2-w7nfz (10.128.1.27)
[student@workstation D0288-apps]$ curl \
http://scale-${RHT_OCP4_DEV_USER}-manage-deploy.${RHT_OCP4_WILDCARD_DOMAIN}
This is version 2 of the app. I am running on host -> scale-2-dxswv (10.129.1.119)
```

Es sollte `version 2` in der Ausgabe angezeigt werden. Sie sollten zudem sehen, dass für die Anforderungen in den zwei Pods eine Round-Robin-Lastverteilung besteht. Beachten Sie, dass sich der Pod-Name und die IP-Adressen von der älteren Bereitstellung unterscheiden.



Anmerkung

Wenn die neue Version nicht angezeigt wird, überprüfen Sie Folgendes:

- Ob die Änderungen an das Remote-Git-Repository übertragen wurden
- Ob der neue Build erfolgreich abgeschlossen wurde

8. Nehmen Sie ein Rollback auf die vorherige Bereitstellung vor.

Führen Sie den Befehl `curl` aus, um die Anwendung erneut zu testen. Verifizieren Sie, dass `version 1` in der Ausgabe angezeigt wird.

- 8.1. Nehmen Sie ein Rollback auf die vorherige Version der Bereitstellung vor.

Es wird eine Warnung angezeigt, dass die Trigger für Image-Änderungen durch den Befehl `oc rollback` deaktiviert wurden:

```
[student@workstation D0288-apps]$ oc rollback dc/scale
deploymentconfig.apps.openshift.io/scale deployment #3 rolled back to scale-1
Warning: the following images triggers were disabled: scale:latest
You can re-enable them with: oc set triggers dc/scale --auto
```

- 8.2. Warten Sie, bis der neue Anwendungs-Pod bereitgestellt wird. Er muss den Status `READY` aufweisen:

```
[student@workstation D0288-apps]$ oc get pods
NAME        READY   STATUS    RESTARTS   AGE
scale-1-build  0/1    Completed  0          40m
scale-1-deploy 0/1    Completed  0          36m
scale-2-build  0/1    Completed  0          10m
scale-2-deploy 0/1    Completed  0          6m59s
scale-3-bcxpath 1/1    Running   0          58s
scale-3-deploy  0/1   Completed  0          77s
scale-3-lnp46  1/1    Running   0          68s
```

- 8.3. Führen Sie den Befehl `curl` aus, um die Anwendungsantwort zu überprüfen.

```
[student@workstation D0288-apps]$ curl \
http://scale-${RHT_OCP4_DEV_USER}-manage-deploy.${RHT_OCP4_WILDCARD_DOMAIN}
This is version 1 of the app. I am running on host -> scale-3-bcxpg (10.128.2.133)
[student@workstation D0288-apps]$ curl \
http://scale-${RHT_OCP4_DEV_USER}-manage-deploy.${RHT_OCP4_WILDCARD_DOMAIN}
This is version 1 of the app. I am running on host -> scale-3-lnp46 (10.131.1.206)
```

Es sollte **version 1** in der Ausgabe angezeigt werden. Sie sollten zudem sehen, dass für die Anforderungen in den zwei Pods eine Round-Robin-Lastverteilung besteht. Beachten Sie, dass sich der Pod-Name und die IP-Adressen von der vorherigen Bereitstellung unterscheiden.

9. Bewerten Sie Ihre Arbeit:

```
[student@workstation D0288-apps]$ lab manage-deploy grade
```

10. Bereinigen und löschen Sie das Projekt.

```
[student@workstation D0288-apps]$ oc delete project \
${RHT_OCP4_DEV_USER}-manage-deploy
```

Beenden

Führen Sie auf **workstation** das Skript `lab manage-deploy finish` aus, um diese Übung zu beenden. Dieser Schritt ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken. Die Aktion „Beenden“ veröffentlicht dieses Projekt und seine Ressourcen.

```
[student@workstation ~]$ lab manage-deploy finish
```

Dadurch wird die Wiederholungsübung abgeschlossen.

Zusammenfassung

In diesem Kapitel wurden die folgenden Themen behandelt:

- Readiness- und Liveness-Probes überwachen den Zustand Ihrer Anwendungen.
- Die Verwendung der **Rolling**-Strategie, wenn Sie gleichzeitig zwei Versionen Ihrer Anwendung ausführen, um ein Upgrade ohne Ausfallzeiten durchzuführen. Diese Strategie skaliert zunächst zusätzliche Pods mit der neuen Version und dann, sobald sie fertig ist, die Pods mit der älteren Version herunter.
- Die Verwendung der **Recreate**-Strategie, wenn es nicht möglich ist, zwei Versionen Ihrer Anwendung gleichzeitig auszuführen. Diese Strategie fährt alle Pods mit der vorherigen Version herunter und startet dann zusätzliche Pods mit der neueren Version.
- Die Verwendung der **Custom**-Strategie, um den Deploymentprozess anzupassen, wenn die zwei von OpenShift bereitgestellten Strategien Ihre Anforderungen nicht erfüllen.
- Die **Recreate**- und **Rolling**-Strategien unterstützen Lifecycle-Hooks. Dadurch können Sie die Bereitstellung an verschiedenen Punkten im Deploymentprozess anpassen.
- Sie können die Ressourcennutzung für Anwendungsbereitstellungen begrenzen, indem Sie Ressourcengrenzen als Teil der Deploymentstrategie angeben.

Kapitel 8

Erstellen von Anwendungen für OpenShift

Ziel

Erstellen und Bereitstellen von Anwendungen auf OpenShift.

Ziele

- Integrieren von containerisierten Anwendungen in nicht containerisierte Services
- Bereitstellen containerisierter Anwendungen von Drittanbietern gemäß den empfohlenen Methoden für OpenShift.
- Verwenden einer Red Hat OpenShift Application Runtime, um eine Anwendung bereitzustellen.

Abschnitte

- Integrieren externer Services (und angeleitete Übung)
- Containerisieren von Services (und angeleitete Übung)
- Bereitstellen einer Cloud-nativen Anwendung mit Eclipse JKube (und angeleitete Übung)

Praktische Übung

Erstellen von Cloud-nativen Anwendungen für OpenShift

Integrieren externer Services

Ziele

In diesem Abschnitt wird beschrieben, wie eine containerisierte Anwendung in nicht containerisierte Services integriert wird.

Überblick zu Red Hat OpenShift-Services

Ein typischer Service in OpenShift weist einen Namen und einen Selektor auf. Ein Service verwendet seinen Selektor, um Pods zu ermitteln, die an den Service gesendete Anwendungsanforderungen empfangen sollten. OpenShift-Anwendungen verwenden den Servicenamen, um eine Verbindung mit den Serviceendpunkten herzustellen.

In ähnlicher Weise ermöglicht ein FQDN einer Anwendung, einen Namen für den Zugriff auf die Endpunkte eines öffentlichen Services zu verwenden. OpenShift-Services ermöglichen jedoch den Anwendungszugriff auf Serviceendpunkte, ohne dass der Service öffentlich zugänglich gemacht werden muss.

Eine Anwendung ermittelt einen Service mit Umgebungsvariablen oder dem internen OpenShift-DNS-Server. Bei Verwendung der Umgebungsvariablen muss der Service definiert werden, bevor der Anwendungs-Pod erstellt wird. Andernfalls empfängt die Anwendung nicht die Umgebungsvariablen.

Die Verwendung des internen OpenShift-DNS-Servers ist flexibler, da Anwendungen Services dynamisch erkennen können. Der Servicename wird für alle Pods im selben OpenShift-Cluster, der den Service enthält, zu einem lokalen DNS-Hostnamen. OpenShift fügt dem DNS-Resolver-Suchpfad sämtlicher Container das Domain-Suffix `svc.cluster.local` hinzu. OpenShift weist zudem dem jeweiligen Service den Hostnamen `service-name.project-name.svc.cluster.local` zu.

Wenn beispielsweise der Service `myapi` im Projekt `myproject` vorhanden ist, können alle Pods im selben OpenShift-Cluster den Hostnamen `myapi.myproject.svc.cluster.local` auflösen, um die Service-IP-Adresse abzurufen. Es sind zudem die folgenden kurzen Hostnamen verfügbar:

- Pods aus demselben Projekt können den Servicenamen `myapi` als einen kurzen Hostnamen ohne Domain-Suffix verwenden.
- Pods aus einem unterschiedlichen Projekt können den Servicenamen und den Projektnamen `myapi.myproject` als einen kurzen Hostnamen ohne Domain-Suffix `svc.cluster.local` verwenden.

Definieren externer Services

OpenShift unterstützt mehrere Ansätze, Services ohne enthaltene Selektoren zu definieren. Dadurch kann ein OpenShift-Service auf mindestens einen Host verweisen, der außerhalb des OpenShift-Clusters liegt.

Angenommen, Sie möchten eine Anwendung containerisieren, die von einem vorhandenen Datenbankservice abhängt, der auf Ihrem OpenShift-Cluster noch nicht verfügbar ist. Sie müssen den Datenbankservice nicht zu OpenShift migrieren, bevor Sie die Anwendung containerisieren. Beginnen Sie stattdessen mit dem Entwerfen ihrer Anwendung für die Interaktion mit OpenShift-

Services, einschließlich des Datenbankservices. Erstellen Sie einfach einen OpenShift-Service, der auf die externen Datenbankserviceendpunkte verweist.

Wenn Sie OpenShift-Services für die Endpunkte externer Services erstellen, können Ihre Anwendungen sowohl interne als auch externe Services ermitteln. Wenn sich die Endpunkte eines externen Services ändern, müssen Sie die betroffenen Anwendungen nicht neu konfigurieren. Aktualisieren Sie stattdessen die Endpunkte für den entsprechenden OpenShift-Service.

Erstellen eines externen Services

Der einfachste Ansatz, einen externen Service zu erstellen, besteht darin, den Befehl `oc create service externalname myservice` mit der Option `--external-name` auszuführen:

```
[user@host ~]$ oc create service externalname myservice \
--external-name myhost.example.com
```

Im vorherigen Beispiel kann auch eine IP-Adresse anstelle eines DNS-Namens akzeptiert werden.

Im OpenShift-Cluster ausgeführte Anwendungen können anschließend den Servicenamen so wie einen normalen Service verwenden, und zwar als eine Umgebungsvariable oder als einen lokalen Hostnamen.

Definieren von Endpunkten für einen Service

Ein typischer Service erstellt anhand des Selektorattributs des Services dynamisch *Endpunkt-Ressourcen*. Die Befehle `oc status` und `oc get all` zeigen diese Ressourcen nicht an. Sie können den Befehl `oc get endpoints` verwenden, um sie anzuzeigen.

Falls Sie den Befehl `oc create service externalname --external-name` ausführen, um einen Service zu erstellen, erstellt der Befehl zudem eine Endpunktressource, die auf den Hostnamen oder die IP-Adresse verweist, die als ein Argument angegeben ist.

Wenn Sie die Option `--external-name` nicht verwenden, wird keine Endpunktressource erstellt. In diesem Fall sollten Sie den Befehl `oc create -f` ausführen und eine Ressourcendefinitionsdatei verwenden, um die Endpunktressourcen explizit zu erstellen.

Wenn Sie einen Endpunkt anhand einer Datei erstellen, können Sie mehrere IP-Adressen für denselben externen Service definieren und auf den Lastverteilungs-Features des OpenShift-Services aufbauen. In diesem Szenario werden von OpenShift weder Adressen hinzugefügt noch entfernt, die für die Verfügbarkeit der jeweiligen Instanz berücksichtigt werden. Eine externe Anwendung muss die Liste der IP-Adressen in der Endpunktressource aktualisieren.

Weitere Informationen zu den Endpunktressourcen-Definitionsdateien finden Sie in den Verweisen am Ende dieses Abschnitts.



Literaturhinweise

Lesen Sie den Abschnitt *Type ExternalName* der Service Concepts-Dokumentation für Kubernetes unter
<https://kubernetes.io/docs/concepts/services-networking/service/#externalname>

Informieren Sie sich über die OpenShift-DNS-Namenskonventionen im Kapitel *Networking* des *Architecture Guide* for Red Hat OpenShift Container Platform 4.6 unter
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/networking/index

Hinweis der Verfasser: Die folgende Ressource ist in der 4.x-Version der Dokumentation für OpenShift noch nicht vorhanden. Die unten verlinkten Informationen sind jedoch nach wie vor für die aktuelle Version 4.5 des Produkts relevant.

Weitere Informationen finden Sie im Kapitel *Integrating External Services* im *Developer Guide* for Red Hat OpenShift Container Platform 3.11 unter
https://docs.openshift.com/container-platform/3.11/dev_guide/integrating_external_services.html

► Angeleitete Übung

Integrieren eines externen Services

In dieser Übung stellen Sie eine Anwendung in OpenShift bereit, die mit einer außerhalb des OpenShift-Clusters befindlichen Datenbank kommuniziert.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Bereitstellen der Anwendung To Do List anhand von Quellcode
- Erstellen eines Datenbankservices für die Anwendung, die auf einen außerhalb des OpenShift-Clusters liegenden MariaDB-Datenbankserver verweist
- Verifizieren, dass die Anwendung die vorab in die Datenbank geladenen Daten verwendet

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen aktiven OpenShift-Cluster
- Auf die Anwendung To Do List im Git-Repository (`todo-single`)
- Auf einen Nexus-Server, der die NPM-Abhängigkeiten bereitstellt, die von der Anwendung (`restify`, `sequelize` und `mysql`) benötigt werden
- Auf das S2I-Builder-Image für Node.js 12
- Auf einen vorbereiteten (vorbestückten) Maria-DB-Datenbankserver, der außerhalb Ihres OpenShift-Clusters ausgeführt wird

Führen Sie den folgenden Befehl auf workstation aus, um die Voraussetzungen zu überprüfen und die Anwendung "To Do List" bereitzustellen:

```
[student@workstation ~]$ lab external-service start
```

Der vorherige Befehl führt das Skript `oc-new-app.sh` im Ordner `~/D0288/labs/external-service` aus, um die Anwendung bereitzustellen. Sie können dieses Skript überprüfen, wenn Sie weitere Informationen zu den Ressourcen der Anwendung To Do List benötigen, die Sie während dieser Übung verwendet haben.

Anweisungen

► 1. Überprüfen Sie die Ressourcen der Anwendung To Do List.

1.1. Laden Sie die Konfiguration Ihrer Kursumgebung.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

1.2. Melden Sie sich bei OpenShift mit Ihrem Entwicklerbenutzerkonto an.

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

13. Geben Sie das Projekt "youruser-external-service" ein, das Ihre Anwendung "To Do List" hostet:

```
[student@workstation ~]$ oc project ${RHT_OCP4_DEV_USER}-external-service
Now using project "youruser-external-service" on server
"https://api.cluster.domain.example.com:6443".
```

14. Verifizieren Sie, dass das OpenShift-Projekt eine einzelne Anwendung mit dem Namen todoapp aufweist, die anhand der Quellen erstellt wurde:

```
[student@workstation ~]$ oc status
...output omitted...
http://todo-youruser-external-service.apps.domain.cluster.example.com to pod port
8080-tcp (svc/todoapp)
dc/todoapp deploys istag/todoapp:latest <-
bc/todoapp source builds https://github.com/youruser/D0288-apps on openshift/
nodejs:12
deployment #1 deployed 26 seconds ago - 1 pod
...output omitted...
```

15. Überprüfen Sie, ob die Anwendung bereit ist und ausgeführt wird:

```
[student@workstation ~]$ oc get pod
NAME          READY   STATUS    RESTARTS   AGE
todoapp-1-6z6qg  1/1     Running   0          1m
todoapp-1-build  0/1     Completed  0          4m
todoapp-1-deploy 0/1     Completed  0          1m
```

16. Überprüfen Sie die Umgebungsvariablen im Anwendungs-Pod, um die Datenbankverbindungsparameter abzurufen.

```
[student@workstation ~]$ oc rsh todoapp-1-6z6qg env | grep DATABASE
DATABASE_PASSWORD=redhat123
DATABASE_SVC=tododb
DATABASE_USER=todoapp
DATABASE_INIT=false
DATABASE_NAME=todo
```

- 2. Verifizieren Sie, dass die Anwendung „To Do List“ nicht in der Lage ist, auf den Datenbankserver tododb zuzugreifen, der von der Umgebungsvariablen DATABASE_SVC angegeben wurde.

- 2.1. Rufen Sie den Hostnamen ab, über den die Anwendung zugänglich ist. Da der Hostname sehr lang ist, speichern Sie ihn in einer Shell-Variablen.

```
[student@workstation ~]$ HOSTNAME=$(oc get route todoapp \
-o jsonpath='{.spec.host}')
[student@workstation ~]$ echo ${HOSTNAME}
todoapp-youruser-external-service.apps.cluster.domain.example.com
```

- 2.2. Führen Sie den Befehl `curl` aus, und verwenden Sie den Hostnamen aus dem vorherigen Schritt sowie den API-Ressourcenpfad `/todo/api/items/6`, um ein Element aus der Datenbank abzurufen. Die Fehlermeldung aus der Anwendung gibt an, dass der Hostname des Services `tododb` nicht aufgelöst werden kann.

```
[student@workstation ~]$ curl -si http://${HOSTNAME}/todo/api/items/6
HTTP/1.1 500 Internal Server Error
...output omitted...
{"message":"getaddrinfo ENOTFOUND tododb tododb:3306"}
```

► 3. Überprüfen Sie die externe Datenbank.

- 3.1. Suchen Sie den Hostnamen des externen MariaDB-Servers. Dieser Hostname entspricht der Platzhalter-Domain Ihres OpenShift-Clusters und ersetzt das Präfix `apps.` durch `mysql.ocp-`.

```
[student@workstation ~]$ dbhost=$(echo \
mysql.ocp-${RHT_OCP4_WILDCARD_DOMAIN#"apps."})
[student@workstation ~]$ echo ${dbhost}
mysql.ocp-cluster.domain.example.com
```

- 3.2. Führen Sie den Befehl `mysqlshow` aus, um auf dem externen MariaDB-Server eine Verbindung zur Datenbank `todo` herzustellen, und verifizieren Sie, dass sie die Tabelle `Item` enthält.

Verwenden Sie den Hostnamen aus dem vorherigen Schritt: Verwenden Sie den Benutzer `todoapp` und das Passwort `redhat123`:

```
[student@workstation ~]$ mysqlshow -h${dbhost} \
-utodoapp -predhat123 todo
Database: todo
+-----+
| Tables |
+-----+
| Item   |
+-----+
```

► 4. Erstellen Sie einen OpenShift-Service, der eine Verbindung zur externen Datenbankinstanz herstellt, und verifizieren Sie, dass die Anwendung nun Daten von der externen Datenbank empfängt.

- 4.1. Führen Sie den Befehl `oc create svc` aus, um einen Service basierend auf einem externen Namen und den Datenbankserver-Hostnamen aus dem vorherigen Schritt zu erstellen.

```
[student@workstation ~]$ oc create svc externalname tododb \
--external-name ${dbhost}
service/tododb created
```

- 4.2. Verifizieren Sie, dass der Service `tododb` vorhanden ist und eine externe IP, aber keine Cluster-IP angezeigt:

```
[student@workstation ~]$ oc get svc
NAME      TYPE        CLUSTER-IP      EXTERNAL-IP      ...
todoapp   ClusterIP   172.30.140.201  <none>          ...
tododb    ExternalName <none>          mysql.ocp-cluster.domain.example.com ...
```

- 4.3. Verifizieren Sie, dass die Anwendung nun Daten von der externen Datenbank empfängt.

Verwenden Sie den Befehl `curl` erneut:

```
[student@workstation ~]$ curl -si  http://${HOSTNAME}/todo/api/items/6
HTTP/1.1 200 OK
...
{"id":6,"description":"Verify that the To Do List application works","done":false}
```

- 5. Führen Sie eine Bereinigung durch. Löschen Sie das Projekt `youruser-external-service` aus OpenShift.

```
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-external-service
```

Beenden

Führen Sie auf der `workstation` den Befehl `lab external-service finish` aus, um diese Übung zu beenden. Dieser Schritt ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken. Die Aktion „Beenden“ veröffentlicht dieses Projekt und seine Ressourcen.

```
[student@workstation ~]$ lab external-service finish
```

Hiermit ist die angeleitete Übung beendet.

Containerisieren von Services

Ziele

In diesem Abschnitt wird beschrieben, wie containerisierte Drittanbieteranwendungen überprüft und bereitgestellt werden, indem die empfohlenen Praktiken für OpenShift befolgt werden.

Überprüfen von containerisierten Anwendungen, die auf OpenShift bereitgestellt werden sollen

Gelegentlich müssen Sie eine Anwendung auf OpenShift bereitstellen, deren Anbieter entweder ein Dockerfile zum Erstellen des Container-Images oder ein vorgefertigtes Container-Image auf einem Registry-Server bereitstellt. Diese Anwendung bietet möglicherweise einen Back-End-Service, der von Ihrer benutzerdefinierten Anwendung benötigt wird, beispielsweise eine Datenbank oder ein Messaging-System oder einen Service, der von Ihrem Entwicklungsprozess benötigt wird, beispielsweise eine gehostete Container-Registry, ein Artefakt-Repository oder ein Collaboration-Tool.

Auch wenn die Anwendung bereits von ihrem Anbieter containerisiert wurde, bedeutet dies nicht unbedingt, dass sie auf OpenShift gut ausgeführt wird. Es gibt mehrere Ursachen, weshalb Ihr erster Versuch fehlschlägt, diese Anwendung mit dem Befehl `oc new-app` bereitzustellen. Zum Beheben dieser Probleme kann es erforderlich sein, die Bereitstellung der Anwendung anzupassen oder das zugehörige Dockerfile zu ändern.

Im Folgenden finden Sie eine Liste mit einigen allgemeinen Problemen:

- Ihre Anwendung wird nicht unter den standardmäßigen OpenShift-Sicherheitsrichtlinien ausgeführt, die durch die eingeschränkten Sicherheitskontextbeschränkungen (SCC) definiert sind. Es wird ggf. erwartet, dass das Container-Image der Anwendung als `root` oder als ein anderer fester Benutzer ausgeführt wird. Möglicherweise sind zudem benutzerdefinierte SELinux-Richtlinien und andere privilegierte Einstellungen erforderlich.

In der Regel können Sie das Dockerfile so ändern, dass die Anwendung die standardmäßigen OpenShift-Sicherheitsrichtlinien erfüllt. Falls nicht, dann bewerten Sie sorgfältig das Risiko der Ausführung der Anwendung unter einer weniger restriktiven Sicherheitsrichtlinie.

- Ihre Anwendung erfordert benutzerdefinierte Einstellungen für Ressourcenanforderungen und Ressourcenlimits. Dies geschieht bei älteren, monolithischen Anwendungen, die nicht für eine Microservice-basierte Architektur entwickelt wurden. OpenShift-Cluster-Administratoren richten Standardgrenzbereiche ein, die Standardwerte für Ressourcenanforderungen und -limits für Ihr Projekt bereitstellen, und diese sind möglicherweise zu klein für Ihre ältere Anwendung.

Sie können erhöhte Ressourcenanforderungen und Ressourcenlimits für Ihre Bereitstellung angeben, um die vom Cluster-Administrator festgelegten Standardwerte zu überschreiten. Wenn Ihre Anwendungsanforderungen das von Ihren OpenShift-Cluster-Administratoren festgelegte Ressourcenkontingent übersteigen, müssen die Administratoren entweder das Ressourcenkontingent für Ihren Benutzer erhöhen oder ein Servicekonto für Ihre Anwendung mit einem erhöhten Ressourcenkontingent bereitstellen.

- Das Anwendungs-Image definiert Konfigurationseinstellungen, die für eine Standalone-Containerlaufzeit in Ordnung sind, jedoch nicht für einen OpenShift- oder einen generischen Kubernetes-Cluster. Anwendungen lesen Konfigurationseinstellungen aus Umgebungsvariablen

und definieren normalerweise Standardwerte mithilfe von ENV-Anweisungen in einem Dockerfile.

Es ist nicht erforderlich, das Dockerfile zu ändern, um ihre Umgebungsvariablen zu ändern. Sie können jede Umgebungsvariable in der Bereitstellung Ihrer Anwendung überschreiben. Dazu zählen auch die, die von den ENV-Anweisungen im Dockerfile nicht explizit festgelegt wurden.

- Ihre Anwendung erfordert persistenten Storage. Anwendungen definierten Storage-Anforderungen in der Regel mithilfe von VOLUME-Anweisungen im Dockerfile. Manchmal erwarten diese Anwendungen, Hostordner für ihre Volumes zu verwenden, was von Ihren OpenShift-Cluster-Administratoren verboten ist. Diese Anwendungen erwarten möglicherweise einen direkten Zugriff auf den Netzwerkspeicher, was aus Sicherheitsaspekten nicht empfohlen wird.

Der richtige Weg, persistenten Storage für Anwendungen in OpenShift bereitzustellen, besteht darin, PVC-Ressourcen (Persistent Volume Claim, Anforderung für ein persistentes Volumes) zu definieren und die Anwendungsbereitstellung zu konfigurieren, um diese PVCs an die Volumes aus dem Anwendungscontainer anzuhängen. OpenShift verwaltet den Netzwerkspeicher im Auftrag Ihrer Anwendungen, und Ihre Cluster-administratoren verwalten Sicherheitsrichtlinien und Leistungsstufen für den Speicher.

Ihr Anbieter stellt möglicherweise Kubernetes-Ressourcendateien zum Bereitstellen von Anwendungen bereit, und diese Ressourcendateien erfordern möglicherweise eine Anpassung, um unter OpenShift zu funktionieren. Deployment-Ressourcen, die mit vorgelagerten Kubernetes (und auch mit anderen Anbieterdistributionen von Kubernetes) arbeiten, funktionieren möglicherweise nicht mit OpenShift, da Standard-Kubernetes nicht mit sicheren Standardeinstellungen ausgestattet ist.

Manchmal stellen Anbieter Kubernetes-Deployment-Ressourcen bereit, vorausgesetzt, der Benutzer ist ein Cluster-Administrator. Diese Anbieter sind möglicherweise nicht der Ansicht, dass ein OpenShift-Cluster in der Regel von mehreren Organisationen und Teams gemeinsam genutzt wird, denen zu ihrer eigenen Sicherheit keine Cluster-Administratorrechte gewährt werden.

Überprüfen des Dockerfiles für die Nexus-Anwendung

Die von Sonatype entwickelte Nexus-Anwendung ist ein Repository-Manager, der häufig von Entwicklern zum Speichern der Artefakte verwendet wird, die von Anwendungen angefordert werden. Dazu zählen beispielsweise Abhängigkeiten. Die Anwendung kann Artefakte für verschiedene Technologien wie Java, .NET, Docker, Node, Python und Ruby speichern.

Sonatype stellt ein Dockerfile bereit, um die Nexus-Anwendung mit einem einfachen Container-Image von Red Hat zu erstellen. Das resultierende Image ist im Red Hat Container Catalog verfügbar, wodurch bestätigt wird, dass es einer Grundmenge an Richtlinien von Red Hat entspricht. Das Dockerfile ist so konfiguriert, dass es die OpenShift-Funktionen unterstützt. Dazu zählt beispielsweise, es Container-Images zu erlauben, als ein zufälliges Benutzerkonto ausgeführt zu werden.

Basis-Image- und Containermetadaten

Das Dockerfile-Projekt ist unter <https://github.com/sonatype/docker-nexus3/tree/3.18.0> verfügbar. Die Datei `Dockerfile.rhel` enthält Folgendes:

```
...output omitted...
FROM      registry.access.redhat.com/rhel7/rhel ①
MAINTAINER Sonatype <cloud-ops@sonatype.com>
```

```
...output omitted...

LABEL name="Nexus Repository Manager" \
    ...output omitted...
    io.k8s.description="The Nexus Repository Manager server \
        with universal support for popular component formats." \
    io.k8s.display-name="Nexus Repository Manager" \
    io.openshift.expose-services="8081:8081" \
    io.openshift.tags="Sonatype,Nexus,Repository Manager"

...output omitted...
```

- ➊ Gibt das Red Hat Enterprise Linux 7-Container-Image als das Basis-Image an. Es gibt auch eine alternative Dockerfile-Datei, die das im GitHub-Projekt befindliche Red Hat Universal Base Image 8 verwendet.
- ➋ Definiert die Image-Metadaten für Kubernetes und OpenShift.

Das Nexus-Dockerfile verwendet zudem die ARG-Anweisung des Dockerfile. Eine ARG-Anweisung definiert eine Variable, die nur während des Image-Build-Prozesses vorhanden ist. Im Gegensatz zu einer ENV-Anweisung sind diese Variablen nicht Teil des resultierenden Container-Images:

```
...output omitted...
ARG NEXUS_VERSION=3.18.0-0-01
ARG NEXUS_DOWNLOAD_URL=https://.../nexus/3/nexus-$NEXUS_VERSION-unix.tar.gz
ARG NEXUS_DOWNLOAD_SHA256_HASH=e1d9...c99e
...output omitted...
```

Der Build-Prozess verwendet diese drei Variablen, um die installierte Version von Nexus im Container-Image zu steuern und zu überprüfen.

Das Dockerfile definiert auch Umgebungsvariablen, die in dem erstellten Container-Image vorhanden sind:

```
...output omitted...
# configure nexus runtime
ENV SONATYPE_DIR=/opt/sonatype
ENV NEXUS_HOME=${SONATYPE_DIR}/nexus \
    NEXUS_DATA=/nexus-data \
    NEXUS_CONTEXT='' \
    SONATYPE_WORK=${SONATYPE_DIR}/sonatype-work \
DOCKER_TYPE='rh-docker'
...output omitted...
```

Der Installationsprozess verwendet die Umgebungsvariable DOCKER_TYPE, um die Installation und Konfiguration anzupassen. Ein Wert von rh-docker ruft die Konfigurationsänderungen auf, die für ein Red Hat-zertifiziertes Container-Image erforderlich sind.

Der Nexus-Installationsprozess

Sonatype bietet Chef-Cookbooks (Kochbücher) an, die die Installation der Nexus-Anwendung standardisieren. Chef ist eine Open-Source-Konfigurationsmanagement-Technologie, die Puppet und Ansible ähnelt und darauf abzielt, den Prozess der Konfiguration und Verwaltung von Servern

Kapitel 8 | Erstellen von Anwendungen für OpenShift

zu optimieren. Ein Chef-Cookbook ist eine Sammlung von Chef-Rezepten, und jedes Rezept definiert die Konfiguration für einen bestimmten Satz von Systemressourcen.

Das Nexus-Dockerfile verwendet den Chef-Installationsprozess, um das Nexus-Container-Image zu erstellen:

```
...output omitted...
ARG NEXUS_REPOSITORY_MANAGER_COOKBOOK_VERSION="release-0.5.20190212-..."①
ARG NEXUS_REPOSITORY_MANAGER_COOKBOOK_URL="https://github.com/sonatype/..."

ADD solo.json.erb /var/chef/solo.json.erb②

# Install using chef-solo
RUN curl -L https://www.getchef.com/chef/install.sh | bash \
&& /opt/chef/embedded/bin/erb /var/chef/solo.json.erb > /var/chef/solo.json \
&& chef-solo \
--node_name nexus_repository_red_hat_docker_build \
--recipe-url ${NEXUS_REPOSITORY_MANAGER_COOKBOOK_URL} \
--json-attributes /var/chef/solo.json \
&& rpm -qa chef | xargs rpm -e \
&& rpm --rebuilddb \
&& rm -rf /etc/chef \
&& rm -rf /opt/chefdk \
&& rm -rf /var/cache/yum \
&& rm -rf /var/chef
...output omitted...
```

- ① Das Dockerfile verwendet Variablen, um die Version des Chef-Cookbooks zu steuern, das Nexus installiert. Die Cookbook-URL wird als Argument für einen Chef-Befehl in einer späteren RUN-Anweisung bereitgestellt.
- ② Im selben Verzeichnis wie das Dockerfile enthält die Datei `solo.json.erb` Chef-Konfigurationsdetails. Das Dockerfile fügt dem Container-Image diese Datei hinzu, um den Chef-Installationsprozess zu aktivieren.
- ③ Die Installation von Nexus erfolgt mit einer einzelnen RUN-Anweisung, die:
 - Chef herunterlädt und installiert.
 - das Chef-Cookbook ausführt, um Nexus mit dem Befehl `chef-solo` zu installieren.
 - heruntergeladene Dateien, Chef-RPMs und andere fremde Dateien entfernt.

Das Chef-Cookbook konfiguriert auch Berechtigungen, sodass `gui=0` in die Daten- und Logordner schreiben kann, wodurch die Standardsicherheitsrichtlinien von OpenShift eingehalten werden.

Containerausführungsumgebung

Der untere Bereich des Dockerfiles enthält Metadaten, mit denen eine Containerlaufzeit einen Container aus dem Image erstellen kann:

```
...output omitted...
VOLUME ${NEXUS_DATA}①

EXPOSE 8081②
```

```
USER nexus③

ENV INSTALL4J_ADD_VM_PARAMS="-Xms1200m -Xmx1200m ..."④

ENTRYPOINT ["/uid_entrypoint.sh"]⑤
CMD ["sh", "-c", "${SONATYPE_DIR}/start-nexus-repository-manager.sh"]⑥
```

- ① Konfiguriert das Verzeichnis /nexus-data als ein Volume, da der Wert der NEXUS_DATA-Variablen /nexus-data lautet. Die Nexus-Anwendung speichert Anwendungsdaten und Ressourcen in diesem Verzeichnis.
- ② Die Nexus-Anwendung kommuniziert über Port 8081.
- ③ Die Nexus-Anwendung wird als der Benutzer nexus ausgeführt. Der Chef-Installationsprozess erstellt und konfiguriert diesen Benutzer.
- ④ Die Nexus-Anwendung funktioniert auf einer Java Virtual Machine (JVM). Die JVM benötigt Optionen, um die Größe ihres Heaps zu ändern, wenn sie auf einer Standalone-Containerlaufzeit ausgeführt wird. Auf OpenShift müssen Sie diese Optionen überschreiben und die JVM OpenShift-Ressourcenlimits und Ressourcenanforderungen einhalten lassen. Andernfalls kann es zu Terminierungs- und Stabilitätsproblemen kommen.
- ⑤ Während der Installation erstellt das Chef-Cookbook das /uid_entrypoint.sh-Skript. Das Skript /uid_entrypoint.sh soll die Benutzer-ID erkennen, welche die Anwendung ausführt, und sie für den Benutzer nexus definieren. Dadurch kann die Anwendung mit standardmäßigen eingeschränkten Sicherheitskontextbeschränkungen normal ausgeführt werden.
- ⑥ Der Befehl, der die Nexus-Anwendung startet. Der Entry Point führt diesen Befehl aus, nachdem er Berechtigungen an den zufällig zugewiesenen Prozess-UID angepasst hat.

Anpassen von OpenShift-Ressourcen für das Nexus-Image

Wenn Sie ein Nexus-Container-Image aus dem offiziellen Dockerfile entwickeln und es mit einem Befehl (z. B. `oc new-app --docker-image`) bereitstellen, funktioniert es möglicherweise nicht zuverlässig. Um sicherzustellen, dass das Container-Image ordnungsgemäß funktioniert, müssen Sie einige Anpassungen an der Bereitstellung vornehmen.

Festlegen von Ressourcenanforderungen und -limits

Red Hat empfiehlt, dass für in OpenShift bereitgestellte Anwendungen Ressourcenanforderungen und Ressourcenlimits festgelegt werden. Wenn Sie keine sehr alten Versionen der Java Virtual Machine (JVM) verwenden, müssen Sie keine JVM-Heap-Optionen definieren.

Ressourcenanforderungen geben die minimale Menge an Ressourcen an, die die Anwendung zur Ausführung benötigt. Der OpenShift-Planer sucht einen Worker-Knoten im Cluster mit ausreichend verfügbarer CPU und Arbeitsspeicher, um die Anwendung auszuführen, wodurch Ressourcenengpässe beim Start verhindert werden.

Ressourcenlimits geben an, wie stark die CPU- und Speicherauslastung einer Anwendung im Laufe der Zeit zunehmen kann. OpenShift legt Linux-Kernel-Cgroups für den Anwendungscontainer fest, was verhindert, dass er diese Limits jemals verletzt. Der Linux-Kernel beendet Container, die diese Limits verletzen, und OpenShift startet Ersatzcontainer, wodurch Probleme durch Speicherverluste, Endlosschleifen und Deadlocks gemildert werden. Die neuesten

Versionen der JVM vergrößern automatisch den Heap und andere interne Datenstrukturen, um die aktuellen Cgroups-Einstellungen nicht zu verletzen.

Sie definieren Ressourcenanforderungen und Ressourcenlimits innerhalb der Pod-Vorlage einer Bereitstellung:

```
...output omitted...
- apiVersion: apps/v1
  kind: Deployment
...output omitted...
  spec:
    ...output omitted...
      template:
        ...output omitted...
          spec:
            containers:
              ...output omitted...
                resources:
                  limits:
                    cpu: "1"
                    memory: 2Gi
                  requests:
                    cpu: 500m
                    memory: 256Mi
...output omitted...
```

Im Nexus-Szenario müssen Sie auch die Umgebungsvariable `INSTALL4J_ADD_VM_PARAMS` überschreiben, um alle JVM-Optionen im Zusammenhang mit der Speichergröße zu entfernen:

```
...output omitted...
- apiVersion: apps/v1
  kind: Deployment
...output omitted...
  spec:
    ...output omitted...
      template:
        ...output omitted...
          spec:
            containers:
              - env:
                  - name: INSTALL4J_ADD_VM_PARAMS
                    value: -Djava.util.prefs.userRoot=/nexus-data/javaprefs
...output omitted...
```

Im vorherigen Beispiel wird davon ausgegangen, dass Sie die Umgebungsvariable `NEXUS_DATA` nicht überschreiben, um einen anderen Volumepfad für den persistenten Storage des Containers anzugeben.

Implementieren von Probes

Red Hat empfiehlt, dass für in OpenShift bereitgestellte Anwendungen Integritätstests festgelegt werden. Nexus bietet eine API, um zu bestimmen, ob der Service funktionsfähig ist und ob er Deadlocks aufweist. Sie können jedoch keinen einfachen HTTP-Probe verwenden, da die Nexus-

API eine Authentifizierung erfordert und die API einen HTTP 200-Statuscode zurückgibt, auch wenn Nexus fehlerhaft ist.

Sie können ein Skript erstellen, das sich mit der Nexus-API authentifiziert und Antworten von der API analysiert. Das folgende Skript verifiziert, dass der Nexus-Service bereit ist, Anforderungen zu verarbeiten:

```
#!/bin/sh
curl -si -u admin:$(cat /nexus-data/admin.password) \
      http://localhost:8081/service/metrics/ping | grep pong
```

- ❶ Die Option `-u` über gibt einen Benutzernamen und ein Passwort in der HTTP GET-Anforderung.
- ❷ Nexus stellt den Endpunkt `/service/metrics/ping` bereit, der den Wert `pong` zurückgeben muss, wenn der Service bereit ist. Wenn `pong` nicht in der Antwort ist, wird der Befehl `grep` mit einem Statuscode ungleich Null beendet.

Der Benutzer `admin` ist berechtigt, Anforderungen an Health Check-Endpunkte zu senden. Wenn die Nexus-Anwendung zum ersten Mal initialisiert wird, generiert sie ein zufälliges Passwort für den Benutzer `admin`. Nexus speichert das zufällige Passwort in der Datei `/nexus-data/admin.password`.

Auf ähnliche Weise bestimmt das folgende Skript, ob der Nexus-Service vorhanden ist:

```
#!/bin/sh
curl -si -u admin:$(cat /nexus-data/admin.password) \
      http://localhost:8081/service/metrics/healthcheck | grep healthy | \
      grep true
```

Da diese Probe-Skripts einfach sind, können Sie den Inhalt jedes Skripts in den entsprechenden Probe-Absatz der Nexus-Containerspezifikation eintragen:

```
...output omitted...
- apiVersion: apps/v1
  kind: Deployment
...output omitted...
  spec:
    ...output omitted...
    template:
      ...output omitted...
      spec:
        containers:
          ...output omitted...
          livenessProbe:
            exec:
              command:
                - /bin/sh ❶
                - "-c" ❷
                - > ❸
                  curl -si -u admin:$(cat /nexus-data/admin.password)
                  http://localhost:8081/service/metrics/healthcheck |
                  grep healthy | grep true
            failureThreshold: 3
```

```
initialDelaySeconds: 10
periodSeconds: 10
...output omitted...
```

- ① Verwenden Sie die Shell /bin/sh.
- ② Die Option -c gibt an, dass ein einzelner Befehl in der Shell ausgeführt wird.
- ③ Der Indikator zur Fortsetzung über mehrere Zeilen in YAML. Auf diese Weise können Sie eine lange Zeichenfolge über mehrere Zeilen aufteilen.

Bei einem großen Probe-Skript sollten Sie das Dockerfile ändern und dem Container-Image die Probe-Skripts hinzufügen.



Literaturhinweise

Weitere Informationen über den Nexus-Service finden Sie in der Dokumentation zu Nexus Repository Manager unter
<https://help.sonatype.com/repomanager3>

Weitere Informationen zu Nexus-Anwendungsintegrität und -Metriken finden Sie unter

Nexus 3 Server Metrics and Health Information – Sonatype Support
<https://support.sonatype.com/hc/en-us/articles/226254487-Nexus-3-Server-Metrics-and-Health-Information>

Weitere Informationen zu JVM-Speichereinstellungen in Containern finden Sie unter

Java inside docker: What you must know to not FAIL
<https://developers.redhat.com/blog/2017/03/14/java-inside-docker/>

► Angeleitete Übung

Containerisieren von Nexus als ein Service

In dieser Übung stellen Sie entsprechend den empfohlenen OpenShift-Praktiken einen Nexus-Server als containerisierte Anwendung bereit.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Erstellen eines Container-Images anhand eines Dockerfiles
- Verifizieren, dass ein Helm-Chart einen Image-Stream verwendet, der ein Container-Image aus einer privaten Registry verwendet
- Verifizieren, dass ein Helm-Chart Ressourcenanforderungen und Ressourcenlimits für eine Java-Anwendung definiert und Umgebungsvariablen überschreibt, die Heap-Größen festlegen würden
- Konfigurieren eines Helm-Charts für die Verwendung einer persistenten Volume-Anforderung
- Konfigurieren eines Helm-Charts für die Verwendung von Liveness- und Readiness-Probes
- Bereitstellen einer Instanz von Nexus mithilfe des Helm-Charts

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen aktiven OpenShift-Cluster
- Auf das Red Hat Universal Base Image 8 (ubi8/ubi)
- Auf die Quelldateien für die Erstellung des Nexus-Container-Images im Git-Repository (nexus3)

Führen Sie den folgenden Befehl auf `workstation` aus, um die Voraussetzungen zu überprüfen und die Dateien herunterzuladen, die zum Durchführen dieser Übung erforderlich sind:

```
[student@workstation ~]$ lab nexus-service start
```

Anweisungen

► 1. Überprüfen und erstellen Sie die Nexus-Datei `Dockerfile`.

- 1.1. Wechseln Sie zum Unterverzeichnis `nexus3` Ihres lokalen Klons des Git-Repositories `DO288-apps`, und checken Sie den `main`-Branch des Kurs-Repositorys aus, um

sicherzustellen, dass Sie diese Übung von einem als fehlerfrei bekannten Zustand aus starten:

```
[student@workstation ~]$ cd ~/D0288-apps/nexus3  
[student@workstation nexus3]$ git checkout main  
...output omitted...
```

- 1.2. Überprüfen Sie die Datei **Dockerfile**. Überprüfen Sie zudem, ob die Nexus-Anwendung ihre Dateien dauerhaft im Ordner `/nexus-data` ablegt:

```
[student@workstation nexus3]$ grep VOLUME Dockerfile  
VOLUME ${NEXUS_DATA}  
[student@workstation nexus3]$ grep NEXUS_DATA Dockerfile  
  NEXUS_DATA=/nexus-data \  
...output omitted...
```

Das Dockerfile definiert die Umgebungsvariable `NEXUS_DATA` mit dem Wert `/nexus-data`. Das Dockerfile verwendet diese Variable, um ein Volume zu definieren, auf dem Nexus Anwendungsdaten speichert.

- 1.3. Überprüfen Sie die Datei **Dockerfile**, und überwachen Sie, ob eine Umgebungsvariable definiert wird, die die Heap-Größen von Java Virtual Machine (JVM) festlegt. Später müssen Sie diese Umgebungsvariable überschreiben, damit die OpenShift-Deploymentkonfiguration die Pod-Speichereinstellungen steuert.

```
[student@workstation nexus3]$ grep ENV Dockerfile  
...output omitted...  
ENV INSTALL4J_ADD_VM_PARAMS="-Xms2703m -Xmx2703m -XX:MaxDirectMemorySize=2703m -  
Djava.util.prefs.userRoot=${NEXUS_DATA}/javaprefs"
```

- 1.4. Laden Sie die Konfiguration Ihrer Kursumgebung.

Führen Sie den folgenden Befehl aus, um die in der ersten angeleiteten Übung erstellten Umgebungsvariablen zu laden:

```
[student@workstation nexus3]$ source /usr/local/etc/ocp4.config
```

- 1.5. Erstellen Sie das Container-Image, und übertragen Sie es auf Ihr persönliches Benutzerkonto bei Quay.io. Sie können die Befehle aus dem Skript `build-nexus-image.sh` unter `~/D0288/labs/nexus-service` ausführen oder kopieren und einfügen.

Ignorieren Sie während des Builds die Warnung "HOSTNAME is not supported for OCI image format".

```
[student@workstation nexus3]$ podman build -t nexus3 .  
...output omitted...  
STEP 33: COMMIT nexus3  
[student@workstation nexus3]$ podman login -u ${RHT_OCP4_QUAY_USER} quay.io  
Password:  
Login Succeeded!  
[student@workstation nexus3]$ skopeo copy \  
containers-storage:localhost/nexus3 \  
docker://quay.io/${RHT_OCP4_QUAY_USER}/nexus3
```

```
...output omitted...
Writing manifest to image destination
Storing signatures
```



Wichtig

Der Buildvorgang sollte nicht länger als 5 Minuten dauern. Während des Tests dieser Übung wurden wesentlich längere Build-Zeiten gemeldet.

Falls Ihr Build nach fünf Minuten nicht abgeschlossen ist, drücken Sie Strg+C, um den Build-Prozess zu beenden. Führen Sie den folgenden Befehl aus, um ein vorgefertigtes Nexus-Image direkt in Ihr persönliches Quay.io-Benutzerkonto zu kopieren und den Build-Prozess zu überspringen:

```
[student@workstation nexus3]$ skopeo copy \
docker://quay.io/redhattraining/nexus3:latest \
docker://quay.io/${RHT_OCP4_QUAY_USER}/nexus3
```

- 2. Schließen Sie das Helm-Chart ab, um Nexus als einen Service bereitzustellen.

Es wird ein teilweises Diagramm zum Bereitstellen von Nexus bereitgestellt. Aktualisieren Sie das Diagramm, um Nexus als einen Service bereitzustellen.

- 2.1. Erstellen Sie eine Kopie des zu bearbeitenden Startdiagramms, und verlassen Sie den Ordner ~/D0288-apps/nexus3.

```
[student@workstation nexus3]$ cp -r ~/D0288/labs/nexus-service/nexus-chart ~/
[student@workstation nexus3]$ cd ~
```

- 2.2. Überprüfen Sie die Datei ~/nexus-chart/templates/imagestream.yaml, um zu verifizieren, dass das aus einem Konfigurationswert festgelegte Container-Image verwendet wird.

```
[student@workstation ~]$ grep -A1 "kind: DockerImage" ~/nexus-chart/templates/
imagestream.yaml
  kind: DockerImage
  name: {{ .Values.imageName }}
```

- 2.3. Öffnen Sie die Datei ~/nexus-chart/values.yaml mit einem Texteditor, und aktualisieren Sie sie, um den Wert imageName so festzulegen, dass er auf das von Ihnen erstellte Container-Image zeigt. Lassen Sie diese Datei im Editor geöffnet.

```
...output omitted...
imageName: quay.io/youruser/nexus3:latest
...output omitted...
```

- 2.4. Überprüfen Sie die Datei ~/nexus-chart/templates/deployment.yaml, um zu verifizieren, dass Ressourcenanforderungen und Ressourcenlimits für den Anwendungs-Pod definiert werden:

```
[student@workstation ~]$ grep -B1 -A5 limits: ~/nexus-chart/templates/deployment.yaml
resources:
  limits:
    cpu: "1"
    memory: {{ .Values.memoryLimit }}
  requests:
    cpu: 500m
    memory: 256Mi
```

- 2.5. Öffnen Sie die Datei ~/nexus-chart/templates/deployment.yaml mit einem Texteditor, und überschreiben Sie dann die Umgebungsvariable INSTALL4J_ADD_VM_PARAMS, um keine JVM-Heap-Größenkonfiguration einzuschließen, und definieren Sie nur die Java-Systemeigenschaften, die für die Anwendung erforderlich sind. Lassen Sie die Datei im Editor geöffnet.

```
apiVersion: v1
kind: Deployment
...output omitted...
  - env:
    - name: INSTALL4J_ADD_VM_PARAMS
      value: -Djava.util.prefs.userRoot=/nexus-data/javaprefs
...output omitted...
```

- 2.6. Aktualisieren Sie die Datei ~/nexus-chart/values.yaml mit Ihrem Texteditor, um den memoryLimit-Wert auf 2703Mi zu setzen.
- 2.7. Die Nexus-Anwendung enthält einen /service/metrics/healthcheck-Endpunkt, der verifiziert, dass die Anwendung vorhanden ist. Der Zugriff auf den Endpunkt erfordert eine Authentifizierung. Wenn die Anwendung gestartet wird, wird das Passwort des Benutzers admin in der Datei /nexus-data/admin.password gespeichert.
Überprüfen Sie den Liveness-Probe in der Deployment-Ressource. Rufen Sie die Datei ~/nexus-chart/templates/deployment.yaml mit einem Texteditor auf, und konfigurieren Sie den Probe so, dass er 120 Sekunden nach dem Start des Containers gestartet wird. Konfigurieren Sie den Probe zudem so, dass die maximale Wartezeit für die Ausführung 30 Sekunden beträgt.

```
apiVersion: v1
kind: Deployment
...output omitted...
  livenessProbe:
    exec:
      command:
        - /bin/sh
        - "-c"
        - '>
          curl -siu admin:$(cat /nexus-data/admin.password)
          http://localhost:8081/service/metrics/healthcheck |
          grep healthy | grep true
...output omitted...
    initialDelaySeconds: 120
```

```
...output omitted...
timeoutSeconds: 30
...output omitted...
```

- 2.8. Die Nexus-Anwendung enthält einen `/service/metrics/ping`-Endpunkt, der verifiziert, dass die Anwendung bereit ist. Der Zugriff auf diesen Endpunkt erfordert zudem eine Authentifizierung.

Überprüfen Sie den Readiness-Probe in der Deployment-Ressource. Konfigurieren Sie den Probe in der Datei `~/nexus-chart/templates/deployment.yaml` so, dass er 120 Sekunden nach dem Start des Containers gestartet wird. Konfigurieren Sie den Probe zudem so, dass die maximale Wartezeit für die Ausführung 30 Sekunden beträgt.

```
apiVersion: v1
kind: Deployment
...output omitted...
  readinessProbe:
    exec:
      command:
        - /bin/sh
        - "-c"
        - '>
          curl -siu admin:$(cat /nexus-data/admin.password)
          http://localhost:8081/service/metrics/ping |
          grep pong
        ...output omitted...
    initialDelaySeconds: 120
    ...output omitted...
    timeoutSeconds: 30
...output omitted...
```

- 2.9. Konfigurieren Sie in der Deployment-Ressource den Volume-Mount so, dass das Volume `nexus-data` auf dem Mount-Pfad von `/nexus-data` verwendet wird:

```
apiVersion: v1
kind: Deployment
...output omitted...
  volumeMounts:
    - mountPath: /nexus-data
      name: nexus-data
...output omitted...
```

- 2.10. Benennen Sie in der Deployment-Ressource das Volume `nexus-data`, und konfigurieren Sie das Volume anschließend so, dass die persistente Volume-Anforderung `nexus-data-pvc` verwendet wird:

```
apiVersion: v1
kind: Deployment
...output omitted...
  volumes:
    - name: nexus-data
      persistentVolumeClaim:
        claimName: nexus-data-pvc
...output omitted...
```

- 2.11. Zum dauerhaften Ablegen von Nexus-Daten ist eine persistente Volume-Anforderung erforderlich. Öffnen Sie die Datei `~/nexus-chart/templates/pvc.yaml` in Ihrem Texteditor. Benennen Sie in der PVC-Ressource den PVC `nexus-data-pvc`, indem Sie das Attribut `metadata.name` aktualisieren:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  labels:
    app: {{ .Values.nexusServiceName }}
  name: nexus-data-pvc
...output omitted...
```

- 2.12. Das Helm-Chart ist nun abgeschlossen. Speichern Sie Ihre Bearbeitungen.

Um die in diesem Schritt vorgenommenen Änderungen zu verifizieren, vergleichen Sie Ihre Helm-Chart-Dateien mit den Lösungsdateien unter `~/D0288/solutions/nexus-service/nexus-chart/`. Wenn Sie sich hinsichtlich Ihrer Bearbeitungen unsicher sind, können Sie die Lösungsdatei kopieren und zum nächsten Schritt wechseln.

- 3. Erstellen Sie ein neues Projekt. Fügen Sie ein Secret hinzu, mit dem jeder Projektbenutzer das Nexus-Container-Image aus Quay.io abrufen kann.

- 3.1. Melden Sie sich bei OpenShift mit Ihrem Entwicklerbenutzerkonto an:

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful
...output omitted...
```

- 3.2. Erstellen Sie ein neues Projekt für die Anwendung.

```
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-nexus-service
Now using project "yourname-nexus-service" on server
"https://api.cluster.domain.example.com:6443".
...output omitted...
```

- 3.3. Verwenden Sie Podman ohne den Befehl sudo, um sich bei Quay.io anzumelden.

```
[student@workstation ~]$ podman login -u ${RHT_OCP4_QUAY_USER} quay.io
Password:
Login Succeeded!
```

- 3.4. Erstellen Sie ein Secret, um auf Ihr persönliches Quay.io-Benutzerkonto zuzugreifen.

```
[student@workstation ~]$ oc create secret generic quayio \
--from-file .dockerconfigjson=${XDG_RUNTIME_DIR}/containers/auth.json \
--type kubernetes.io/dockerconfigjson
secret/quayio created
```

- 3.5. Verknüpfen Sie das Secret mit dem Servicekonto default.

```
[student@workstation ~]$ oc secrets link default quayio --for pull
```

- 3.6. Legen Sie den richtigen Hostnamen in ~/nexus-chart/values.yaml fest.
Kopieren Sie die Ausgabe des folgenden Befehls, und fügen Sie sie in die Wertedatei ein, um den Wert hostname festzulegen.

```
[student@workstation ~]$ echo "nexus3-${RHT_OCP4_DEV_USER}.\n${RHT_OCP4_WILDCARD_DOMAIN}"
```

► **4.** Erstellen Sie eine neue Anwendung.

- 4.1. Erstellen Sie eine neue Anwendung mit dem Namen nexus3 aus dem Helm-Chart.
Sie können den vollständigen Befehl kopieren oder direkt über /home/student/DO288/labs/nexus-service/oc-new-app.sh ausführen.

```
[student@workstation ~]$ helm install nexus3 ~/nexus-chart
NAME: nexus3
LAST DEPLOYED: Mon May 31 12:05:20 2021
NAMESPACE: youruser-nexus-service
STATUS: deployed
REVISION: 1
NOTES:
..output omitted...
```

- 4.2. Warten Sie, bis der Anwendungs-Pod ausgeführt wird, aber nicht bereit ist. Verfolgen Sie die Pod-Logs, um die langwierige Initialisierungsprozedur des Nexus-Servers zu beobachten. Wenn die Meldung "Started" angezeigt wird, können Sie mit der Logbeobachtung aufhören.

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
nexus3-1-kfwwh  0/1     Running      0          1m25s
[student@workstation ~]$ oc logs -f nexus3-1-kfwwh
...output omitted...
-----
Started Sonatype Nexus OSS 3.30.1-01
-----
...output omitted...
```

Drücken Sie Ctrl+C, um den Befehl oc logs zu verlassen.

- 4.3. Warten Sie, bis die Anwendung bereit ist und ausgeführt wird. OpenShift benötigt ggf. einige Sekunden, um einen fehlerfreien Zustand aus den Integritäts-Probes der Anwendung zu erhalten.

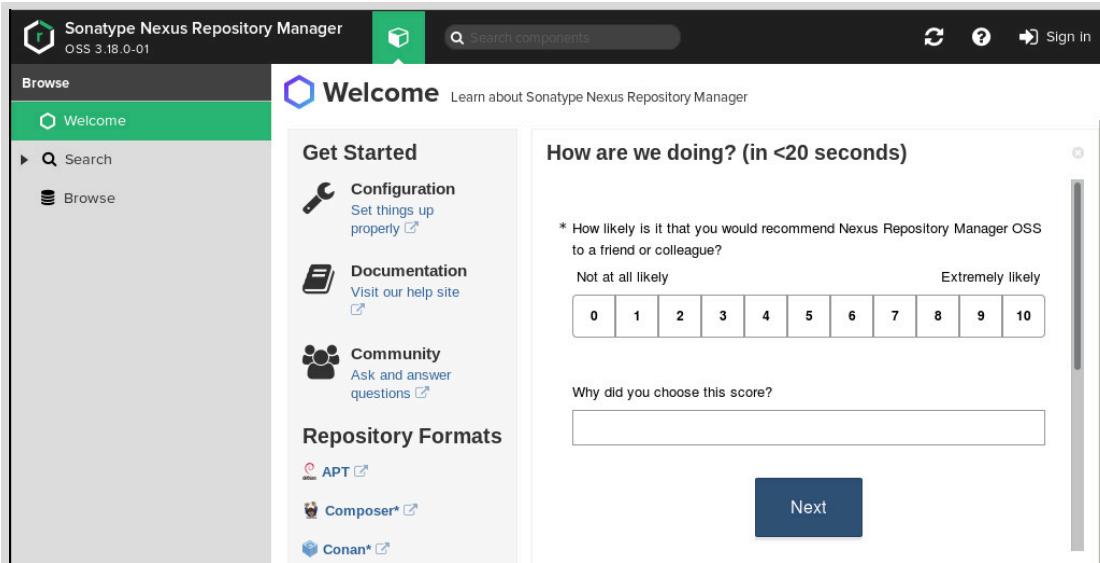
```
[student@workstation ~]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
nexus3-a1b2c3d4e5f6-kfwwh   1/1     Running   0          4m25s
```

► 5. Testen Sie die Nexus-Anwendung.

- 5.1. Rufen Sie die Route für die Nexus-Anwendung ab:

```
[student@workstation ~]$ oc get route
NAME      HOST/PORT   ...
nexus3   nexus3-yourname.apps.cluster.domain.example.com ...
```

- 5.2. Öffnen Sie einen Webbrowser, und navigieren Sie zur Anwendungsroute. Auf dieser Seite wird die Nexus-Anwendung angezeigt.



Falls Sie sich als der Benutzer `admin` anmelden möchten, müssen Sie das Passwort aus der Datei `/nexus-data/admin.password` im Container abrufen. Es ist nicht erforderlich, sich im Rahmen dieses Tests anzumelden.

► 6. Löschen Sie das Projekt in OpenShift und den Container und das Image in der externen Container-Registry. Da Quay.io die Wiederherstellung alter Container-Images ermöglicht, müssen Sie Ihr Repository auf Quay.io ebenfalls löschen.

- 6.1. Löschen Sie das OpenShift-Projekt:

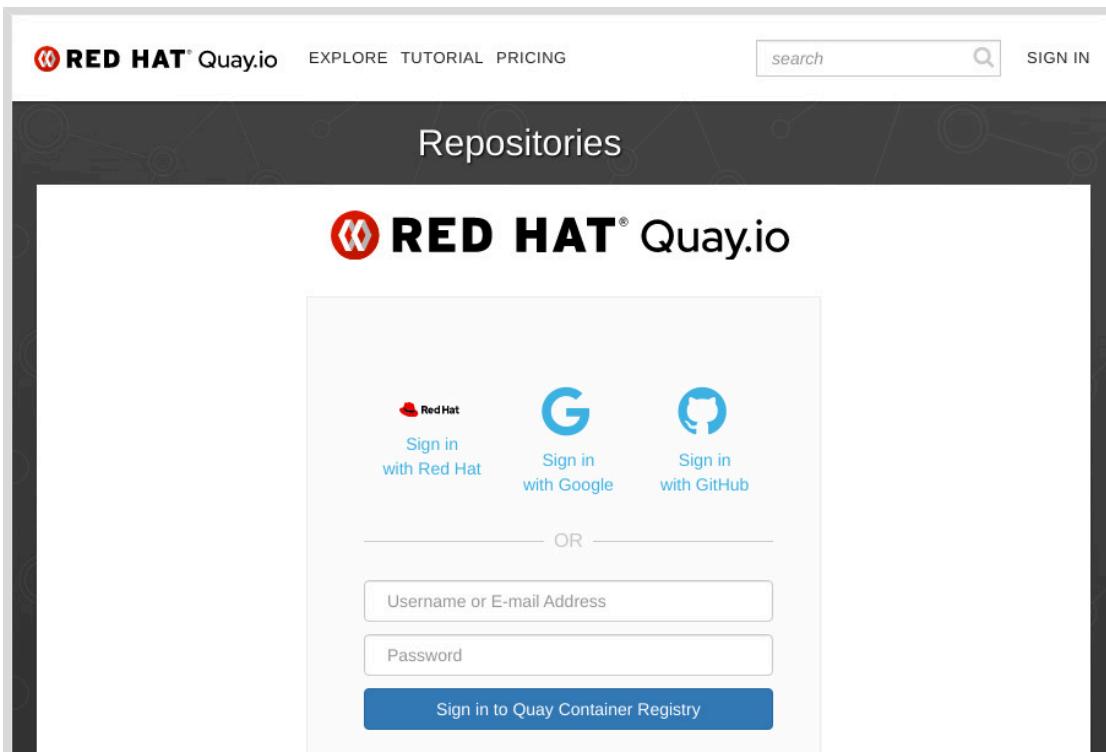
```
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-nexus-service
project.project.openshift.io "youruser.nexus-service" deleted
```

- 6.2. Löschen Sie das Container-Image aus der externen Registry:

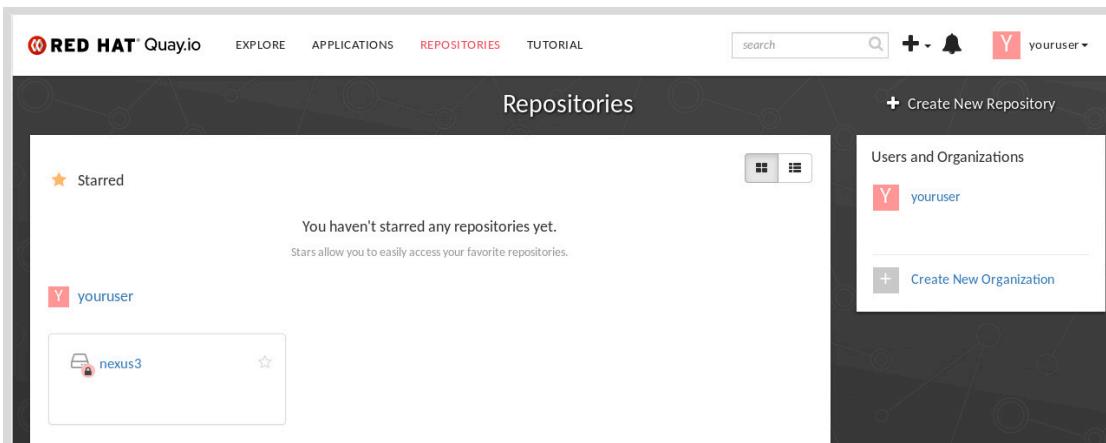
```
[student@workstation ~]$ skopeo delete \
docker://quay.io/${RHT_OCP4_QUAY_USER}/nexus3
```

- 6.3. Melden Sie sich bei Quay.io mit Ihrem persönlichen kostenlosen Benutzerkonto an.

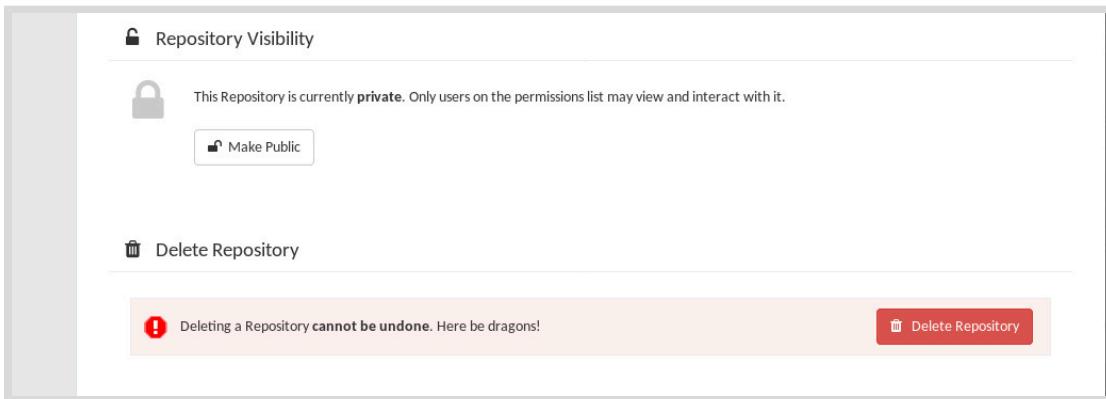
Navigieren Sie zu <https://quay.io> und klicken Sie dann auf **Sign In**, um Ihre Benutzeranmeldedaten einzugeben. Klicken Sie auf **Sign in to Quay Container Registry**, um sich bei Quay.io anzumelden.



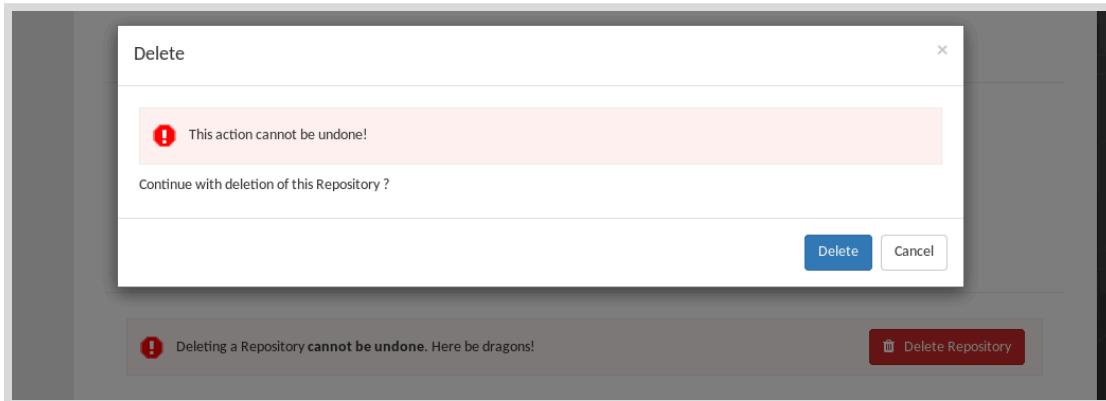
- 6.4. Klicken Sie im Hauptmenü von Quay.io auf **Repositories**, und suchen Sie nach **nexus**. Das Schlosssymbol zeigt an, dass es sich um ein privates Repository handelt, das eine Authentifizierung für Abrufe und Übertragungen erfordert. Klicken Sie auf **nexus3**, um die Seite **Repository Activity** anzuzeigen.



- 6.5. Scrollen Sie auf der Seite **Repository Activity** für das nexus3-Repository nach unten und klicken Sie auf das Zahnradsymbol, um den den **Settings**-Tab anzuzeigen. Scrollen Sie im Tab **Settings** nach unten, und klicken Sie auf **Delete Repository**.



- 6.6. Klicken Sie im Dialogfeld **Delete** auf **Delete**, um das Löschen des nexus3-Repositorys zu bestätigen. Nach einigen Augenblicken werden Sie zur Seite **Repositories** zurückgeleitet. Melden Sie sich bei Quay.io ab.



Beenden

Führen Sie auf der **workstation** den Befehl **lab nexus-service finish** aus, um diese Übung zu beenden. Dieser Schritt ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken. Die Aktion „Beenden“ veröffentlicht dieses Projekt und seine Ressourcen.

```
[student@workstation ~]$ lab nexus-service finish
```

Hiermit ist die angeleitete Übung beendet.

Bereitstellen Cloud-nativer Anwendungen mit JKube

Ziele

In diesem Abschnitt wird das Bereitstellen einer Cloud-nativen Anwendung mit Eclipse JKube behandelt.

Cloud-native Anwendungen

„Cloud-nativ“ ist ein sehr allgemeiner Begriff, mit dem Technologien beschrieben werden sollen. Diese Technologien wurden entwickelt, um skalierbare Anwendungen in Public, Private und Hybrid Clouds zu erstellen und auszuführen, wobei Container, Microservices, Kubernetes und andere moderne Technologien als Beispiele dienen.

Cloud-native Technologien bieten Resilienz, Wartbarkeit und Beobachtbarkeit, und ihre Prozesse werden automatisiert, um häufige Updates und Bereitstellungen zu ermöglichen. So müssen Entwickler, die Cloud-native Tools wie Quarkus oder JKube verwenden, zur Erstellung ihrer Container-Images keine Dockerfiles manuell erstellen.

Eine auf OpenShift bereitgestellte Anwendung, die die von der Plattform bereitgestellten Services verwenden soll, gilt als eine Cloud-native Anwendung.

Eclipse JKube

Eclipse JKube besteht aus einer Reihe von Open Source-Plug-Ins und -Bibliotheken, die Container-Images mit verschiedenen Strategien erstellen können. Java-Anwendungen werden teilweise ganz ohne Konfiguration in Kubernetes und OpenShift generiert und bereitgestellt, wodurch Ihre Anwendungen Cloud-nativ werden.

JKube ist das Ergebnis der Umgestaltung und Umbenennung von Fabric8, die zur Entkopplung des Fabric8-Ökosystems geführt haben. Jetzt ist es ein allgemeineres Plug-in für Anwendungen für Kubernetes und OpenShift.

JKube besteht aus dem JKube Kit, dem wichtigsten Satz an Tools zum Erstellen von Images und Generieren von Deploymentmanifesten, dem Kubernetes Maven-Plug-In und dem OpenShift Maven-Plug-In. Diese Plug-Ins werden für die Bereitstellung im entsprechenden Cluster-Typ verwendet.

JKube unterstützt über seine Maven-Plug-Ins verschiedene Java-Frameworks, beispielsweise:

Quarkus

Ein modernes Cloud-natives Full-Stack-Framework mit geringem Speicherbedarf und kürzerer Startzeit.

Spring Boot

Ein Cloud-natives Entwicklungs-Framework, das auf dem beliebten Spring Framework und der automatischen Konfiguration basiert.

Vert.x

Ein reaktives Entwicklungs-Framework mit niedriger Latenz, das auf asynchronen E/A- und Ereignis-Streams basiert.

Micronaut

Ein modernes Full-Stack-Toolkit zur Erstellung von modularen, einfach zu testenden Microservices und serverlosen Anwendungen.

Open Liberty

Eine flexible Serverlaufzeit für Java-Anwendungen.

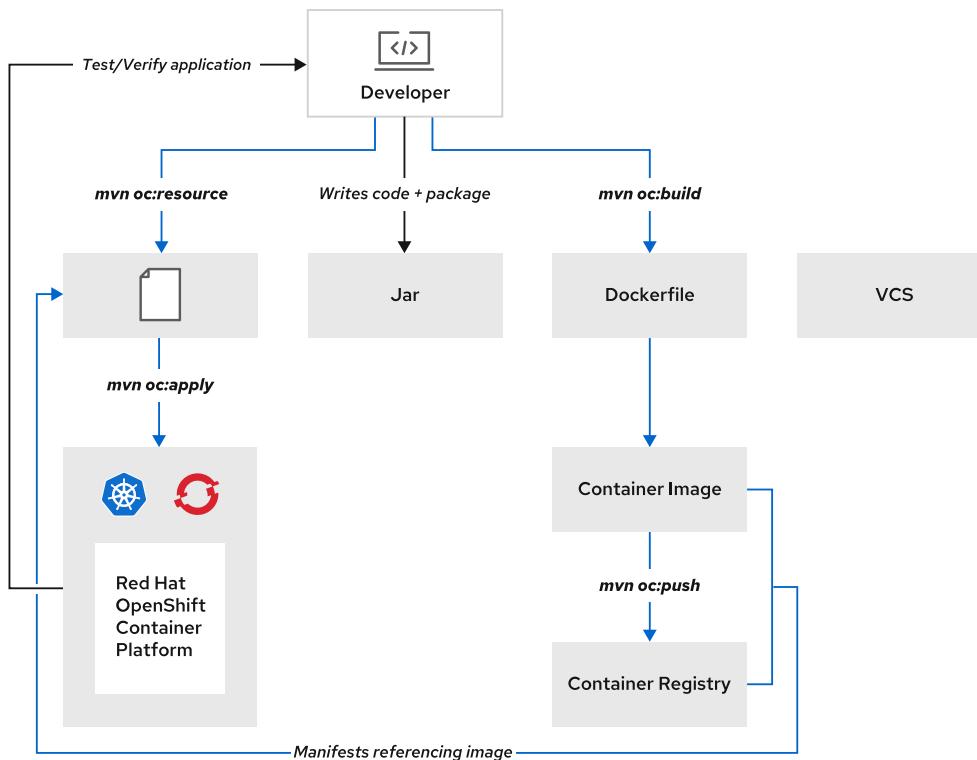
Developer Workflow mit JKube

Abbildung 8.6: Entwickler-Workflow von Eclipse JKube.

Die Verwendung von JKube wird Teil des Entwickler-Workflows. Die unterschiedlichen Ziele helfen dem Entwickler, Ressourcendeskriptoren zu erstellen, sie im OpenShift-Cluster bereitzustellen, Container-Images zu erstellen und sie an eine gewünschte Registry zu übertragen.

Es gibt weitere Funktionen in den Maven-Plug-Ins von JKube, die nicht in diesen Abschnitt fallen. Dadurch kann der Entwickler seinen Workflow verbessern.

Mit diesem Workflow kann der Entwickler eine Bereitstellung ohne Konfiguration in einer Entwicklungsumgebung vornehmen sowie Inline-Konfigurationsbereitstellungen und externe Konfigurationsbereitstellungen für echte Deploymentdeskriptoren.

Kubernetes Maven-Plug-In

Das Kubernetes Maven-Plug-In von JKube ermöglicht Entwicklern das Generieren von Container-Images und Deploymentdeskriptoren sowie das Konfigurieren von Bereitstellungen.

OpenShift Maven-Plug-In

Das JKube OpenShift Maven-Plug-In, das auf dem Kubernetes Maven-Plug-In basiert, ist eine der Kernkomponenten des Eclipse JKube-Open-Source-Projekts.

Sie können das Plug-in verwenden, um Java-Anwendungen mithilfe einer binären Build-Eingabequelle in einem OpenShift-Cluster bereitzustellen. Dies bedeutet, dass das Plug-in die Anwendung kompiliert und verpackt, das Container-Image mit dem verpackten Artefakt generiert und OpenShift-Ressourcen erstellt, um die Anwendung auf OpenShift bereitzustellen.

Konfiguration des OpenShift Maven-Plug-Ins

Um das JKube OpenShift Maven-Plug-In mit einem Projekt zu verwenden, aktualisieren Sie die Datei `pom.xml` des Projekts, um das Plug-In zu aktivieren und zu konfigurieren. Sie müssen der Auflistung `plugins` im Abschnitt `build` der `pom.xml` einen `plugin`-Eintrag hinzufügen. Die folgende Konfiguration ist eine minimale Konfiguration, um das JKube OpenShift Maven-Plug-In für eine Java-Anwendung zu aktivieren:

```
...output omitted...
<build>
  <plugins>
    <plugin>❶
      <groupId>org.eclipse.jkube</groupId>
      <artifactId>openshift-maven-plugin</artifactId>
      <version>1.2.0</version>
      <executions>❷
        <execution>
          <goals>
            <goal>resource</goal>
            <goal>build</goal>
            <goal>apply</goal>
          </goals>
        </execution>
      </executions>
      <configuration>❸
        <!-- additional configuration here -->
      </configuration>
    </plugin>
  </plugins>
</build>
```

- ❶ Stellt Maven die erforderlichen Informationen bereit, damit das Plug-In gefunden, heruntergeladen und verwendet werden kann
- ❷ Konfiguriert das standardmäßige `install`-Ziel für Maven. Das ist vollkommen optional.
- ❸ Zusätzliche Konfiguration, die spezifischer sein kann, sodass Sie Images, Ressourcen, Volumes für die Replikatsätze, Services und ConfigMaps detailliert konfigurieren können

Fügen Sie der Datei `pom.xml` Ihres Projekts dieses XML-Segment hinzu, um das JKube OpenShift Maven-Plug-In hinzuzufügen. Weitere Informationen zur Konfiguration des JKube OpenShift Maven-Plug-Ins finden Sie am Ende der Lektion unter „Referenzen“.

Es gibt auch Unterstützung für Eigenschaften, die mehrere Ziele ändern. Mit ihnen können Sie Standardverhalten ändern, z. B. das Erzwingen der Neuerstellung einer Ressource, sobald Sie

`oc:apply` ausführen, das Erzwingen der Verwendung von `Deployment`-Objekten anstatt der Verwendung von `DeploymentConfig`, und das Ändern anderer Standardverhalten. Weitere Informationen zu diesen Eigenschaften finden Sie auf der Referenzseite am Ende des Abschnitts.

OpenShift Maven-Plug-in Ziele

Ein Maven-Plug-In-Ziel stellt eine klar definierte Aufgabe im Software Development Lifecycle dar. Sie führen ein Maven-Plug-in-Ziel mit dem Befehl `mvn` aus:

```
[user@host sample-app]$ mvn <plug-in goal name>
```

Das OpenShift Maven-Plug-in bietet eine Reihe von Zielen für die Entwicklung von Cloud-nativen Java-Anwendungen:

`oc:resource`

Erstellt Kubernetes- und OpenShift-Ressourcendeskriptoren. Das Plug-in speichert alle generierten Deskriptoren im Unterverzeichnis `target/classes/META-INF/openshift` des Projekts.

`oc:build`

Kompliert und verpackt die Java-Anwendung, um ein binäres Artefakt zu erstellen, und verwendet dann das Artefakt, um das zugehörige Anwendungscontainer-Image zu erstellen.

Das Plug-in verwendet Generatoren, um Build-Parameter automatisch zu erkennen, die zum Komplizieren und Verpacken der Anwendung erforderlich sind. Von besonderer Bedeutung ist, dass Generatoren ein geeignetes Builder-Image identifizieren, das für die Anwendung verwendet werden soll. Bei generischen Java-Anwendungen ist das standardmäßige Builder-Image `quay.io/jkube/jkube-java-binary-s2i`.

Wenn das Plug-in den Zugriff auf einen OpenShift-Cluster erkennt, dann initiiert es einen OpenShift-Binär-Build. Das Plug-in unterstützt sowohl Source-To-Image- als auch Docker-Binär-Builds. Standardmäßig führt das Plug-in eine Source-To-Image-Build-Strategie aus.

Bei OpenShift-Builds erstellt das Plug-in eine Anwendungs-Build-Konfiguration und Image-Stream-Ressource im OpenShift-Cluster. Sowohl für Quell- als auch für Docker-Buildstrategien aktualisiert das Plug-in den Image-Stream mit dem neuen Container-Image.

`oc:apply`

Wendet die generierten Ressourcen auf einen verbundenen OpenShift-Cluster an.

`oc:deploy`

Ähnlich wie `oc:apply`, mit der Ausnahme, dass es im Hintergrund ausgeführt wird.

`oc:undeploy`

Entfernt Ressourcen aus dem OpenShift-Cluster.

`oc:watch`

Überwacht Dateien auf Änderungen, die dann eine Neuerstellung und erneute Bereitstellung auslösen. Dies ist bei der Entwicklung nützlich.

Das JKube OpenShift Maven-Plug-in unterstützt andere Ziele, die über den Rahmen dieses Kurses hinausgehen. Weitere Informationen zu den Zielen finden Sie am Ende der Lektion unter "Referenzen".



Wichtig

Das package-Ziel muss in den meisten Fällen weiterhin vor den zuvor genannten oc-Zielen ausgeführt werden. Dies bedeutet, dass Sie ein entsprechendes Build-Plug-In benötigen, das Ihr Framework oder Ihre Laufzeit unterstützt, um das package zu erstellen. Anschließend können Sie die Images, Ressourcen und die Bereitstellung mithilfe von JKube erstellen.

Anpassen von OpenShift-Ressourcen

Mit minimaler Projektkonfiguration generiert das JKube OpenShift Maven-Plug-in eine Reihe von OpenShift-Ressourcen, um die Bereitstellung Ihrer Anwendung auf OpenShift zu unterstützen. In einigen Szenarien reichen die generierten OpenShift-Ressourcen möglicherweise nicht für die OpenShift-Bereitstellung aus.

Sie können die generierten Ressourcen auf zwei Arten anpassen: Fügen Sie dem Unterverzeichnis `src/main/jkube` des Projekts YAML-Fragmente der OpenShift-Ressource hinzu, oder fügen Sie der Datei `pom.xml` des Projekts eine zusätzliche Konfiguration hinzu. In diesem Kurs verwenden Sie YAML-Ressourcenfragmente, um die OpenShift-Konfiguration für ein Projekt anzupassen.

Wenn Sie Ihrem Projekt nur die minimale JKube-Konfiguration hinzufügen, erstellt das Plug-in eine Service- und Deploymentkonfigurationsressource für Ihre Anwendung. Wenn das zugeordnete Container-Image einen Port zugänglich macht, erstellt das Plug-in auch eine Routenressource, um den Service zugänglich zu machen. Das Plug-in schreibt jede Ressourcendefinition in eine Datei im Verzeichnis `target/classes/META-INF/jkube/openshift`. Alle diese Ressourcendefinitionen werden in einer `openshift.yml`-Datei im Unterverzeichnis `target/classes/META-INF/jkube` für das Projekt zusammengefasst.

Das Plug-in verarbeitet auch YAML-Fragmentdateien im Verzeichnis `src/main/jkube` und verwendet den Inhalt in jedem Fragment, um die entsprechende Standardressourcendefinition zu überschreiben. Der Inhalt jeder Datei imitiert die Struktur einer OpenShift-Ressource, lässt jedoch alle Informationen aus, die unverändert sind. Diese Dateien sind klein und kompakt und stellen nur die Konfiguration dar, die gegenüber der Standard-Ressourcenkonfiguration geändert werden muss.

Jede Fragmentdatei folgt einer Dateibenennungskonvention. Das Plug-in verwendet den Dateinamen des Fragments, um die zu überschreibende OpenShift-Ressource zu identifizieren. Fragmentdateinamen folgen dem Muster `[name] -type.yml`.

Der Wert `name` ist optional und stellt den Namen der Ressource dar. Falls kein Name angegeben ist, verwendet das Plug-in den Anwendungsnamen als Namen der Ressource.

Der Wert `type` entspricht der Art der OpenShift-Ressource, die dieses Fragment ändert. Der Wert "type" muss einem der im Folgenden angegebenen entsprechen:

Art	Dateiname
Service	<code>svc, service</code>
Route	<code>route</code>
Bereitstellung	<code>deployment</code>
DeploymentConfig	<code>dc, deploymentconfig</code>

ConfigMap	cm, configmap
Secret	secret

Sehen Sie sich beispielsweise den folgenden Inhalt von `src/main/jkube/route.yml` an:

```
spec:  
  host: app.alternate.com
```

Dieses Fragment ändert den Standardhostnamen für die Anwendungsroute in `app.alternate.com`.

Sie können auch ein ConfigMap-Objekt mit der Datei `src/main/jkube/cm.yml` erstellen. Beispiel:

```
apiVersion: v1  
kind: ConfigMap  
metadata:  
  name: example-data  
data:  
  MSG_TEXT: This is a text value
```



Literaturhinweise

Die Open-Source-Dokumentation für das JKube OpenShift Maven-Plug-in ist verfügbar unter
<https://www.eclipse.org/jkube/docs/openshift-maven-plugin>

► Angeleitete Übung

Bereitstellen einer Anwendung mit JKube

In dieser Übung verwenden Sie das Eclipse JKube Maven-Plug-in, um einen Microservice auf dem OpenShift-Cluster bereitzustellen.

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Konfigurieren einer benutzerdefinierten OpenShift-Deployment-Ressource mit dem Eclipse JKube Maven-Plug-in
- Konfigurieren einer OpenShift ConfigMap-Ressource mit dem Eclipse JKube Maven-Plug-in
- Bereitstellen eines Microservices mit dem Eclipse JKube Maven-Plug-in

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen aktiven OpenShift-Cluster
- Auf die Microservice-Anwendung im Git-Repository (`micro-java`)

Führen Sie den folgenden Befehl auf `workstation` aus, um die Voraussetzungen zu überprüfen und die Dateien herunterzuladen, die zum Durchführen dieser Übung erforderlich sind:

```
[student@workstation ~]$ lab micro-java start
```

Anweisungen

► 1. Bereiten Sie Ihre Kursumgebung vor.

1. Laden Sie die Konfiguration Ihrer Kursumgebung. Führen Sie den folgenden Befehl aus, um die in der ersten angeleiteten Übung erstellten Umgebungsvariablen zu laden:

```
[student@workstation micro-java]$ source /usr/local/etc/ocp4.config
```

2. Melden Sie sich bei OpenShift mit Ihrem Entwicklerbenutzernamen an.

```
[student@workstation micro-java]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

3. Erstellen Sie ein neues Projekt für die Anwendung. Stellen Sie dem Projektnamen Ihren Entwicklerbenutzernamen voran.

```
[student@workstation micro-java]$ oc new-project ${RHT_OCP4_DEV_USER}-micro-java
Now using project "youruser-micro-java" on server ...
...output omitted...
```

- 2. Untersuchen Sie den Java-Quellcode der Beispielanwendung `micro-java`. Bei der Anwendung handelt es sich um einen Java Quarkus-Microservice, der in einer Nicht-Kubernetes-Umgebung bereitgestellt werden kann. Sie fügen der Anwendung die erforderliche Konfiguration hinzu, um eine Bereitstellung in einem OpenShift Container Platform-Cluster durchzuführen.
- 2.1. Geben Sie das Unterverzeichnis `micro-java` Ihres lokalen Klons des Git-Repositorys `D0288-apps` ein. Checken Sie den `main`-Branch des Kurs-Repositorys aus, um sicherzustellen, dass Sie diese Übung von einem als fehlerfrei bekannten Zustand aus starten:

```
[student@workstation ~]$ cd ~/D0288-apps/micro-java
[student@workstation micro-java]$ git checkout main
...output omitted...
```

- 2.2. Erstellen Sie einen neuen Branch, um alle Änderungen zu speichern, die Sie während dieser Übung vornehmen:

```
[student@workstation micro-java]$ git checkout -b micro-config
Switched to a new branch 'micro-config'
[student@workstation micro-java]$ git push -u origin micro-config
...output omitted...
 * [new branch]      micro-config -> micro-config
Branch micro-config set up to track remote branch micro-config from origin.
```

- 2.3. Überprüfen Sie die Quellcodedatei `HelloResource.java` im Unterverzeichnis `src/main/java/com/redhat/training/openshift/hello` des Quellcodes `micro-java`:

```
package com.redhat.training.openshift.hello;

import java.util.Optional;

import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

import org.eclipse.microprofile.config.inject.ConfigProperty;

@Path("/api")
public class HelloResource {❶

    @ConfigProperty(name = "HOSTNAME", defaultValue = "unknown")
    String hostname;
    @ConfigProperty(name = "APP_MSG")❷
    Optional<String> message;
```

```

    @GET
    @Path("/hello") ③
    @Produces("text/plain")
    public String hello() {
        String response = "";

        if (!message.isPresent()) {
            response = "Hello world from host " + hostname + "\n"; ④
        } else {
            response = "Hello world from host [" + hostname + "].\n";
            response += "Message received = " + message + "\n"; ⑤
        }
        return response;
    }
}

```

- ① Diese Klasse definiert eine REST-Ressource für die Anwendung.
 - ② Die Anwendung verwendet die Umgebungsvariablen `HOSTNAME` und `APP_MSG`, die über `ConfigProperty`-Annotationen injiziert werden.
 - ③ Da Sie unter `/api` auf die Ressource zugreifen, greifen Sie auf diesen Endpunkt unter `/api/hello` zu.
 - ④ Wenn die Umgebungsvariable `APP_MSG` nicht definiert ist, antwortet die Anwendungsressource mit einer Meldung, die den Hostnamen des Servers enthält, auf dem die Anwendung ausgeführt wird.
 - ⑤ Wenn die Umgebungsvariable `APP_MSG` definiert ist, enthält die Antwortmeldung den Wert der beiden Umgebungsvariablen `HOSTNAME` und `APP_MSG`.
- 3. Konfigurieren Sie die Anwendung mithilfe des JKube-Plug-ins für die Bereitstellung in OpenShift. Überprüfen Sie die Konfiguration des JKube Maven-Plug-ins in der Projektdatei `pom.xml`:
- 3.1. Aktualisieren Sie die Attribute der Datei `pom.xml` auf Projektebene, die sich oben befinden, sowie die JKube-Konfigurationsdateien. Legen Sie die Projekteigenschaft `jkube.build.switchToDeployment` auf `true` fest. Dies ist erforderlich, da JKube standardmäßig `DeploymentConfig`-Objekte mit OpenShift verwendet. Wir möchten jedoch stattdessen `Deployment`-Objekte verwenden.

```

<?xml version="1.0" encoding="UTF-8"?>
...output omitted...
<groupId>com.redhat.training.openshift</groupId>
<artifactId>micro-java</artifactId> ①
<version>1.0</version> ②
...output omitted...
<properties>
...output omitted...
    <jkube.build.switchToDeployment>true</jkube.build.switchToDeployment>
</properties>
...output omitted...

```

- ❶ Der Name und die Version der Anwendung. Das JKube Maven-Plug-in erstellt aus diesen Werten die Image-Stream-Tag-Ressource `micro-java:1.0`.
 - ❷ Diese Versionseigenschaften legen die Version der Quarkus-Anwendung fest.
- 3.2. Überprüfen Sie die Liste der Plug-ins im Abschnitt `build` in der Datei `pom.xml` nahe dem Ende der Datei, und fügen Sie am Ende des Builds das folgende Plug-in hinzu:

```
...output omitted...
<build>
  <plugins>
...output omitted...
  <!-- JKube Maven plugin -->
  <plugin>
    <groupId>org.eclipse.jkube</groupId>❶
    <artifactId>openshift-maven-plugin</artifactId>
    <version>1.2.0</version>
  </plugin>
...output omitted...
```

- ❶ Wir verwenden das Plug-in `openshift-maven-plugin` für die Entwicklung und Bereitstellung in OpenShift.
- 3.3. Öffnen Sie die Datei `src/main/jkube/cm.yml`. Diese Datei beschreibt das ConfigMap-Objekt `configmap-hello`, das während der Bereitstellung im OpenShift-Projekt erstellt wird.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configmap-hello
data:
```

- 3.4. Öffnen Sie die Datei `src/main/jkube/deployment.yml`. Diese Datei überschreibt einige von JKube verwendeten Standardwerte. Fügen Sie die Eigenschaft `envFrom` für den Container hinzu, mit der Sie die Umgebungsvariablen aus dem ConfigMap `configmap-hello` festlegen können. Durch Hinzufügen dieses Verweises zum ConfigMap stehen der Anwendung die definierten Einträge als Umgebungsvariablen zur Verfügung.

```
...output omitted...
spec:
  containers:
  - env:
    - name: KUBERNETES_NAMESPACE
      valueFrom:
        fieldRef:
          fieldPath: metadata.namespace
    envFrom:
    - configMapRef:
        name: configmap-hello
  image: micro-java:1.0
```

```
imagePullPolicy: IfNotPresent
name: quarkus
...output omitted...
```

► 4. Generieren Sie OpenShift-Ressourcen für die Anwendung.

- 4.1. Führen Sie im Projektverzeichnis das `mvn package oc:build oc:resource` aus. Das `package` erstellt eine JAR-Datei, die von JKube für die Bereitstellung verwendet werden kann. `oc:build` nimmt in OpenShift einen s2i-Build vor und erstellt ein Image mit dem Namen `micro-java`, das für das OpenShift-Projekt verfügbar ist. Die `oc:resource` erstellt mehrere Ressourcendateien im YAML-Format, die die Anwendung für die Bereitstellung vorbereiten:

```
[student@workstation micro-java]$ mvn -DskipTests package oc:build oc:resource
...output omitted...
[INFO] -----
[INFO] Building jar: /home/student/D0288-apps/micro-java/target/micro-
java-1.0.jar①
[INFO] -----
...output omitted...
[INFO] --- openshift-maven-plugin:1.2.0:build (default-cli) @ micro-java - ②
[INFO] oc: Using OpenShift build with strategy S2I
[INFO] oc: Running in OpenShift mode
[INFO] oc: Running generator quarkus
[INFO] oc: quarkus: Using Docker image quay.io/jkube/jkube-java-binary-s2i:0.0.9
as base / builder
[INFO] oc: [micro-java:latest] "quarkus": Created docker source tar /home/student/
D0288-apps/micro-java/target/docker/micro-java/latest/tmp/docker-build.tar
[INFO] oc: Updating BuildServiceConfig micro-java-s2i for Source strategy
[INFO] oc: Adding to ImageStream micro-java
[INFO] oc: Starting Build micro-java-s2i
[INFO] oc: Waiting for build micro-java-s2i-2 to complete...
...output omitted...
[INFO] --- openshift-maven-plugin:1.2.0:resource (default-cli) @ micro-java - ③
[INFO] oc: Using docker image name of namespace: your-project
[INFO] oc: Running generator quarkus
[INFO] oc: quarkus: Using Docker image quay.io/jkube/jkube-java-binary-s2i:0.0.9
as base / builder
[INFO] oc: Using resource templates from /home/student/D0288-apps/micro-java/src/
main/jkube
[INFO] oc: jkube-service: Adding a default service 'micro-java' with ports [8080]
...output omitted...
```

- ① Das `package`-Ziel erzeugt eine jar-Datei, die für die restlichen Schritte erforderlich ist. Sie könnten auch einen nativen Build haben, der zu einer nativen ausführbaren Datei führt.
- ② Mit dem Ziel `oc:build` wird die Anwendung containerisiert, indem ein Image erstellt wird.
- ③ Das Maven-Ziel `oc:resource` erstellt drei YAML-Dateien. Diese Dateien definieren zusammen einen Service, eine Bereitstellung und eine Routenressource für die Beispielanwendung. Diese Dateien befinden sich im

Unterverzeichnis target/classes/META-INF/jkube/openshift des Projekts.

- 4.2. Überprüfen Sie die generierte Datei micro-java-deployment.yaml im Verzeichnis target/classes/META-INF/jkube/openshift.

```
apiVersion: apps.openshift.io/v1
kind: Deployment
metadata:
...output omitted...
  name: micro-java
spec:
  replicas: 1
...output omitted...
  template:
    spec:
      containers:
        - env:
            - name: KUBERNETES_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
            envFrom:
              - configMapRef:
                  name: configmap-hello
          image: micro-java:1.0
...output omitted...
```

Die Datei definiert eine Bereitstellung mit dem Namen `micro-java`, die einen einzelnen Container mit einem Image aus dem Image-Stream-Tag `micro-java:1.0` bereitstellt. Sie verwendet auch die ConfigMap, wie in der Datei `src/main/jkube/deployment.yaml` angegeben.

- 4.3. Überprüfen Sie die generierte Datei micro-java-service.yaml im Verzeichnis target/classes/META-INF/jkube/openshift.

```
...
apiVersion: v1
kind: Service
metadata:
...output omitted...
  name: micro-java
spec:
  ports:
    - name: http
      port: 8080
      protocol: TCP
      targetPort: 8080
  selector:
    app: micro-java
    provider: jkube
    group: com.redhat.training.openshift
```

Diese Datei definiert einen Service mit dem Namen `micro-java` für die Anwendungs-Pods, die in der Bereitstellung `micro-java` definiert sind.

- 4.4. Überprüfen Sie die generierte Datei `micro-java-route.yml` im Verzeichnis `target/classes/META-INF/jkube/openshift`.

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
...output omitted...
name: micro-java
spec:
port:
  targetPort: 8080
to:
  kind: Service
  name: micro-java
```

Diese Datei definiert eine Routenressource mit dem Namen `micro-java`, die den Service `micro-java` zugänglich macht.

- 4.5. Das Maven-Ziel `oc:resource` generiert zudem eine Liste sämtlicher Ressourcen. Überprüfen Sie die generierte Datei `openshift.yml` im Unterverzeichnis `target/classes/META-INF/jkube` des Projekts:

```
...output omitted...
kind: "List"
...output omitted...
kind: "Service"
...output omitted...
kind: "Deployment"
...output omitted...
kind: "Route"
...output omitted...
```

► 5. Erzeugen Sie die Anwendung und stellen Sie sie bereit.

- 5.1. Erstellen und stellen Sie die Anwendung mit dem JKube Maven-Plug-in bereit. Überwachen Sie die Ausgabe, um zu verifizieren, dass der Build im OpenShift-Modus ausgeführt wird und dass die OpenShift-Ressourcen erstellt wurden.

```
[student@workstation micro-java]$ mvn -DskipTests oc:deploy
...output omitted...
[INFO] >>> openshift-maven-plugin:1.2.0:deploy (default-cli) > install @ micro-
java >>
[INFO]
...output omitted...
[INFO] --- openshift-maven-plugin:1.2.0:deploy (default-cli) @ micro-java -①
[INFO] oc: Using OpenShift at https://api.cluster.domain.example.com:6443 in
namespace your-project with manifest /home/student/D0288-apps/micro-java/target/
classes/META-INF/jkube/openshift.yml ②
[INFO] oc: OpenShift platform detected
[INFO] oc: Creating a Service from openshift.yml namespace your-project name
micro-java③
[INFO] oc: Created Service: target/jkube/applyJson/your-project/service-micro-
java.json
```

```
[INFO] oc: Creating a ConfigMap from openshift.yml namespace your-project name configmap-hello
[INFO] oc: Created ConfigMap: target/jkube/applyJson/your-project/configmap-configmap-hello.json
[INFO] oc: Creating a Deployment from openshift.yml namespace your-project name micro-java
[INFO] oc: Created Deployment: target/jkube/applyJson/your-project/deployment-micro-java.json
[INFO] oc: Creating Route your-project:micro-java host: null
[INFO] oc: HINT: Use the command oc get pods -w to watch your pods start up
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 20.921 s ④
...output omitted...
```

- ❶ Die Deploymentphase beginnt.
- ❷ Das Plug-in erkennt das aktive OpenShift-Projekt. Alle Ressourcen werden in diesem Projekt erstellt.
- ❸ Das Plug-in verwendet die `openshift.yml` zum Erstellen eines OpenShift-Services, einer Bereitstellung und einer Routenressource.
- ❹ Die Build-Zeit sollte innerhalb einer Minute betragen.

Sie sollten in der Lage sein, sofort mit der Aktivität fortzufahren, wenn der Befehl erfolgreich abgeschlossen wird.

► 6. Überprüfen Sie die vom JKube Maven-Plug-in erstellten Ressourcen.

```
[student@workstation micro-java]$ oc status
In project youruser-micro-java on server ...

http://micro-java-youruser... to pod port 8080 (svc/micro-java)
  deployment/micro-java deploys ...
    deployment #1 deployed 2 minutes ago - 1 pod
  bc/micro-java-s2i source builds uploaded code on quay.io/jkube/jkube-java-binary-s2i:0.0.9
    -> istag/micro-java:1.0
...output omitted...
```

In Ihrem Projekt sind eine Routen-, Service- und Deployment-Ressource vorhanden, die jeweils den Namen `micro-java` aufweisen. Eine Build-Konfiguration und eine Image-Stream-Tag-Ressource sind ebenfalls im Projekt vorhanden.

► 7. Testen Sie die Anwendung.

- 7.1. Warten Sie, bis die neuen Anwendungs-Pod bereitgestellt werden. Wenn die Ausgabe von `oc get pods -w` angibt, dass eine neue Bereitstellung bereit ist und ausgeführt wird, fahren Sie mit dem nächsten Schritt fort.

```
[student@workstation micro-java]$ oc get pods -w
NAME                  READY   STATUS    RESTARTS   AGE
micro-java-a1b2c3d4e5f6-5pw6q   1/1     Running   1          14m
micro-java-s2i-1-build         0/1     Completed  0          16m
```

- 7.2. Testen Sie den externen Zugriff auf die Anwendung. Beachten Sie, dass Sie am Ende der Routen-URL /api/hello hinzufügen, um auf die HelloResource-REST-Ressource für die Anwendung zuzugreifen.

```
[student@workstation D0288-apps]$ ROUTE_URL=$(oc get route \
micro-java --template='{{.spec.host}}')
[student@workstation D0288-apps]$ curl ${ROUTE_URL}/api/hello
Hello world from host micro-java-1-5pw6q
```

Da die Anwendungsbereitstellung keinen Wert für die Umgebungsvariable APP_MSG definiert, enthält die Ausgabe nur den Namen des Container-Hosts.

- 8. Aktualisieren Sie das Projekt, um Anwendungs-Pods mit einem neuen Wert für die Umgebungsvariable APP_MSG bereitzustellen.
- 8.1. Aktualisieren Sie die Fragmentdatei src/main/jkube/cm.yml, um einen neuen APP_MSG-Wert für sie bereitzustellen.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configmap-hello
data:
  APP_MSG: this is a new value
```

Das vorhergehende YAML-Fragment definiert eine ConfigMap-Ressource mit dem Namen configmap-hello. Diese ConfigMap definiert eine APP_MSG-Variable mit einem Wert der externen Beispielkonfiguration.

- 8.2. Committen und übertragen Sie die YAML-Fragmente an den Branch micro-config:

```
[student@workstation micro-java]$ git add src/main/jkube/*.yml
[student@workstation micro-java]$ git commit -am "Add YAML fragments."
[student@workstation micro-java]$ git push
...output omitted...
```

- 9. Stellen Sie die Anwendung erneut bereit. Verifizieren Sie, dass ein JKube Maven-Plug-in eine ConfigMap-Ressource erstellt. Verifizieren Sie, dass die Anwendung mit dem Wert der APP_MSG-Variablen aus der ConfigMap antwortet.
- 9.1. Stellen Sie die Anwendung mit dem JKube Maven-Plug-in erneut bereit:

```
[student@workstation micro-java]$ mvn -DskipTests oc:build oc:resource oc:apply  
...output omitted...  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
...output omitted...
```

9.2. Verifizieren Sie das Vorhandensein der configmap-hello-ConfigMap:

```
[student@workstation micro-java]$ oc get cm/configmap-hello  
NAME          DATA   AGE  
configmap-hello    1     84m
```

9.3. Warten Sie, bis die neuen Anwendungs-Pod bereitgestellt werden. Verwenden Sie den Befehl `oc get pods -w`, um wie in einem vorherigen Schritt zu warten.

```
[student@workstation micro-java]$ oc get pods -w  
NAME           READY   STATUS    RESTARTS   AGE  
micro-java-1a2b3c4d5e6f-n2rn2   1/1     Running   0          108s  
micro-java-s2i-1-build   0/1     Completed   0          15m
```

9.4. Verifizieren Sie, dass die Anwendungsantwort den neuen Wert der APP_MSG-Variablen enthält:

```
[student@workstation micro-java]$ curl ${ROUTE_URL}/api/hello  
Hello world from host [micro-java-3-m9k4j].  
Message received = this is a new value
```



Anmerkung

Wenn der vorherige Befehl `curl` eine fehlerhafte HTML-Seite zurückgibt, die besagt, dass die Anwendung nicht verfügbar ist, bedeutet dies, dass der Anwendungscontainer noch keine Anforderungen verarbeiten kann. Warten Sie einige Sekunden, und versuchen Sie es mit demselben Befehl erneut.

- 10. Führen Sie eine Bereinigung durch: Löschen Sie alle während dieser Übung erstellten Ressourcen.

```
[student@workstation micro-java]$ oc delete project \${RHT_OCP4_DEV_USER}-micro-java
```

Beenden

Führen Sie auf `workstation` das Skript `lab micro-java finish` aus, um diese Übung zu beenden. Dieser Schritt ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken. Die Aktion „Beenden“ veröffentlicht dieses Projekt und seine Ressourcen.

```
[student@workstation ~]$ lab micro-java finish
```

Hiermit ist die angeleitete Übung beendet.

► Praktische Übung

Erstellen von Cloud-nativen Anwendungen für OpenShift

In diesem Lab verwenden Sie das Eclipse JKube Maven-Plug-in, um eine Anwendung in OpenShift bereitzustellen, die mit einer außerhalb des OpenShift-Clusters befindlichen Datenbank kommuniziert

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Erstellen eines Datenbankservices für die Anwendung, die auf einen außerhalb des OpenShift-Clusters liegenden MariaDB-Datenbankserver verweist
- Konfigurieren von benutzerdefinierten OpenShift-Ressourcen mithilfe von JKube-YAML-Fragmenten
- Bereitstellen der Back-End-Anwendung `To Do List` mit dem JKube Maven-Plug-in

Bevor Sie Beginnen

Zum Durchführen dieser Aufgabe benötigen Sie Zugriff:

- Auf einen aktiven OpenShift-Cluster
- Auf die Beispielanwendung `todo-api` im Git-Repository
- Auf den externen MariaDB-Datenbankserver

Führen Sie den folgenden Befehl auf `workstation` aus, um die Voraussetzungen zu überprüfen, die Datenbank aufzufüllen und die Dateien herunterzuladen, die zum Durchführen dieser Übung erforderlich sind:

```
[student@workstation ~]$ lab todo-migrate start
```

Anforderungen

Das Entwicklungsteam für das `To Do List`-Back-End migriert die Thorntail-Anwendung zu OpenShift. Der Anwendungsquellcode befindet sich im Unterverzeichnis `todo-api` des geklonten Git-Repositorys.

Entsprechend den folgenden Anforderungen müssen Sie die Anwendung konfigurieren und sie auf einem OpenShift-Cluster bereitstellen:

- Der Projektname lautet `youruser-todo-migrate`. Ihr Benutzer "developer" muss der Inhaber des Projekts sein.
- In Ihrem OpenShift-Projekt ist ein `tododb`-Service vorhanden, der eine Verbindung mit einer externen Datenbank herstellt. Um auf den FQDN der externen Datenbank zuzugreifen, ersetzen Sie `apps` in der Platzhalter-Domain Ihres OpenShift-Clusters durch `mysql.ocp-`. Beispiel: Ist die Platzhalter-Domain des Clusters `apps.cluster.domain.example.com`, lautet der FQDN `mysql.ocp-cluster.domain.example.com`.

- Eine OpenShift ConfigMap-Ressource wird als Teil der Bereitstellung erstellt.

Die ConfigMap definiert Variablen, die für den Zugriff auf die externe Datenbank erforderlich sind:

Verwenden Sie ein JKube YAML-Fragment, um die ConfigMap-Ressource zu erstellen.

- **DATABASE_USER:** todoapp
- **DATABASE_PASSWORD:** redhat123
- **DATABASE_SVC_HOSTNAME:** tododb
- **DATABASE_NAME:** todo

- Eine benutzerdefinierte OpenShift-Deployment-Ressource wird als Teil der Bereitstellung erstellt.

Verwenden Sie ein JKube YAML-Fragment, um alle Variablen aus der ConfigMap-Ressource als Umgebungsvariablen in den Anwendungscontainer einzufügen.

- Committen Sie alle Codeänderungen an das Remote-Git-Repository.

Um eine erfolgreiche Bereitstellung zu testen, sollte der Anwendungsendpunkt `todo/api/items/6` JSON-Daten zurückgeben.

Anweisungen

1. Verifizieren Sie die Konnektivität zum externen Datenbankserver.
2. Erstellen Sie einen OpenShift-Service mit dem Namen `tododb`, der eine Verbindung zur externen Datenbankinstanz herstellt. Der Service muss im Projekt `youruser-todo-migrate` erstellt werden.
3. Erstellen Sie einen `todo-migrate`-Branch anhand des `master`-Branches in Ihrem lokalen Klon des DO288-apps-Repositorys. Wechseln Sie in das Unterverzeichnis `todo-api` des Projekts.
4. Erzeugen Sie die Anwendung und stellen Sie sie bereit. Verifizieren Sie, dass der Anwendungs-Pod nicht bereitgestellt werden kann, da die erforderlichen Umgebungsvariablen nicht definiert sind.
5. Erstellen Sie das YAML-Fragment, das JKube zum Generieren der benutzerdefinierten ConfigMap-Ressource verwendet, die die Datenbankumgebungsvariablen definiert.
Sie können die erforderlichen Umgebungsvariablen in einer benutzerdefinierten ConfigMap-Ressource oder direkt in der Deploymentkonfiguration definieren.
6. Erstellen Sie das YAML-Fragment, das von JKube verwendet wird, um die benutzerdefinierte Deploymentkonfigurationsressource zu generieren.
7. Wenden Sie die benutzerdefinierte ConfigMap und die Deploymentkonfigurationsressource auf das Projekt an. Verifizieren Sie, dass die Anwendung ohne Fehler bereitgestellt wird.
Verwenden Sie die externe Route, um den Zugriff auf die Anwendungsressource `todo/api/items/6` zu testen. Eine erfolgreiche Antwort gibt JSON-Daten zurück.
8. Nach erfolgreichen Anwendungstests committen und übertragen Sie Ihre Codeänderungen an das Remote-Repository.

Bewertung

Verwenden Sie als Benutzer `student` auf dem Rechner `workstation` den Befehl `lab`, um Ihre Arbeit zu bewerten. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie den Befehl so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab todo-migrate grade
```

Beenden

Führen Sie auf der `workstation` den Befehl `lab todo-migrate finish` aus, um diese Übung zu beenden. Dieser Schritt ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken.

```
[student@workstation ~]$ lab todo-migrate finish
```

Hiermit ist die praktische Übung abgeschlossen.

► Lösung

Erstellen von Cloud-nativen Anwendungen für OpenShift

In diesem Lab verwenden Sie das Eclipse JKube Maven-Plug-in, um eine Anwendung in OpenShift bereitzustellen, die mit einer außerhalb des OpenShift-Clusters befindlichen Datenbank kommuniziert

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Erstellen eines Datenbankservices für die Anwendung, die auf einen außerhalb des OpenShift-Clusters liegenden MariaDB-Datenbankserver verweist
- Konfigurieren von benutzerdefinierten OpenShift-Ressourcen mithilfe von JKube-YAML-Fragmenten
- Bereitstellen der Back-End-Anwendung `To Do List` mit dem JKube Maven-Plug-in

Bevor Sie Beginnen

Zum Durchführen dieser Aufgabe benötigen Sie Zugriff:

- Auf einen aktiven OpenShift-Cluster
- Auf die Beispielanwendung `todo-api` im Git-Repository
- Auf den externen MariaDB-Datenbankserver

Führen Sie den folgenden Befehl auf `workstation` aus, um die Voraussetzungen zu überprüfen, die Datenbank aufzufüllen und die Dateien herunterzuladen, die zum Durchführen dieser Übung erforderlich sind:

```
[student@workstation ~]$ lab todo-migrate start
```

Anforderungen

Das Entwicklungsteam für das `To Do List`-Back-End migriert die Thorntail-Anwendung zu OpenShift. Der Anwendungsquellcode befindet sich im Unterverzeichnis `todo-api` des geklonnten Git-Repositorys.

Entsprechend den folgenden Anforderungen müssen Sie die Anwendung konfigurieren und sie auf einem OpenShift-Cluster bereitstellen:

- Der Projektname lautet `youruser-todo-migrate`. Ihr Benutzer "developer" muss der Inhaber des Projekts sein.
- In Ihrem OpenShift-Projekt ist ein `tododb`-Service vorhanden, der eine Verbindung mit einer externen Datenbank herstellt. Um auf den FQDN der externen Datenbank zuzugreifen, ersetzen Sie `apps` in der Platzhalter-Domain Ihres OpenShift-Clusters durch `mysql.ocp-`. Beispiel: Ist die Platzhalter-Domain des Clusters `apps.cluster.domain.example.com`, lautet der FQDN `mysql.ocp-cluster.domain.example.com`.

Kapitel 8 | Erstellen von Anwendungen für OpenShift

- Eine OpenShift ConfigMap-Ressource wird als Teil der Bereitstellung erstellt.

Die ConfigMap definiert Variablen, die für den Zugriff auf die externe Datenbank erforderlich sind:

Verwenden Sie ein JKube YAML-Fragment, um die ConfigMap-Ressource zu erstellen.

- DATABASE_USER: todoapp
- DATABASE_PASSWORD: redhat123
- DATABASE_SVC_HOSTNAME: tododb
- DATABASE_NAME: todo
 - Eine benutzerdefinierte OpenShift-Deployment-Ressource wird als Teil der Bereitstellung erstellt.

Verwenden Sie ein JKube YAML-Fragment, um alle Variablen aus der ConfigMap-Ressource als Umgebungsvariablen in den Anwendungskontainer einzufügen.

- Committen Sie alle Codeänderungen an das Remote-Git-Repository.

Um eine erfolgreiche Bereitstellung zu testen, sollte der Anwendungspunkt `todo/api/items/6` JSON-Daten zurückgeben.

Anweisungen

1. Verifizieren Sie die Konnektivität zum externen Datenbankserver.

- 1.1. Laden Sie die Konfiguration Ihrer Kursumgebung.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

- 1.2. Bestimmen Sie den Hostnamen des externen Datenbankservers.

```
[student@workstation ~]$ MYSQL_DB=$(echo \
mysql.ocp-$RHT_OCP4_WILDCARD_DOMAIN#"apps.")
```

- 1.3. Stellen Sie eine Verbindung zur externen MySQL-Datenbank her.

```
[student@workstation ~]$ mysql -h${MYSQL_DB} -utodoapp -predhat123 todo
Reading table information for completion of table and column names
...output omitted...
mysql>
```

- 1.4. Beenden Sie den MySQL-Client, um zum Shell-Prompt zurückzukehren.

```
mysql> exit
Bye
[student@workstation ~]$
```

Kapitel 8 | Erstellen von Anwendungen für OpenShift

2. Erstellen Sie einen OpenShift-Service mit dem Namen `tododb`, der eine Verbindung zur externen Datenbankinstanz herstellt. Der Service muss im Projekt `youruser-todo-migrate` erstellt werden.

- 2.1. Melden Sie sich bei OpenShift mit Ihrem Entwicklerbenutzerkonto an:

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful
...output omitted...
```

- 2.2. Erstellen Sie ein neues Projekt zum Hosten der Anwendung:

```
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-todo-migrate
```

- 2.3. Erstellen Sie einen Service, der auf einem externen Namen basiert:

```
[student@workstation ~]$ oc create service externalname tododb \
--external-name ${MYSQL_DB}
service "tododb" created
```

- 2.4. Verifizieren Sie, dass der Service `tododb` vorhanden ist und eine externe IP, aber keine Cluster-IP anzeigt:

```
[student@workstation ~]$ oc get svc
NAME      TYPE           ...   EXTERNAL-IP           PORT(S)    AGE
tododb   ExternalName   ...   mysql.cluster.domain.example.com <none>     6s
```

3. Erstellen Sie einen `todo-migrate`-Branch anhand des `master`-Branches in Ihrem lokalen Klon des `D0288-apps`-Repositorys. Wechseln Sie in das Unterverzeichnis `todo-api` des Projekts.

```
[student@workstation ~]$ cd ~/D0288-apps
[student@workstation D0288-apps]$ git checkout main
...output omitted...
[student@workstation D0288-apps]$ git checkout -b todo-migrate
Switched to a new branch 'todo-migrate'
[student@workstation D0288-apps]$ git push -u origin todo-migrate
...output omitted...
[student@workstation D0288-apps]$ cd todo-api
[student@workstation todo-api]$
```

4. Erzeugen Sie die Anwendung und stellen Sie sie bereit. Verifizieren Sie, dass der Anwendungs-Pod nicht bereitgestellt werden kann, da die erforderlichen Umgebungsvariablen nicht definiert sind.

- 4.1. Kompilieren und erzeugen Sie die Anwendung, und stellen Sie sie bereit.

```
[student@workstation todo-api]$ mvn clean compile package oc:build oc:resource
oc:apply
```

- 4.2. Überwachen Sie nach der Bereitstellung der Anwendung die Logs für den Anwendungs-Pod.

```
[student@workstation todo-api]$ oc get pods
NAME           READY   STATUS      RESTARTS   AGE
todo-api-558555647-mpg9j   0/1     CrashLoopBackOff   3          101s
todo-api-s2i-1-build       0/1     Completed   0          2m34s
```

Der genaue STATUS des Pods kann variieren, aber die Anzahl der NEUSTARTS erhöht sich.



Anmerkung

Der STATUS des Pods kann Running lauten, bevor er den Status CrashLoopBackOff erreicht.

```
[student@workstation todo-api]$ oc logs -f todo-api-558555647-mpg9j
...output omitted...
One or more configuration errors have prevented the application from starting. The
errors are:
- SRCFG00011: Could not expand value DATABASE_USER in property
quarkus.datasource.username
- SRCFG00011: Could not expand value DATABASE_PASSWORD in property
quarkus.datasource.password
- SRCFG00011: Could not expand value DATABASE_SVC_HOSTNAME in property
quarkus.datasource.jdbc.url
```

Es tritt ein Fehler auf, da unter anderem die Umgebungsvariablen DATABASE_SVC_HOSTNAME und DATABASE_NAME nicht definiert sind.

5. Erstellen Sie das YAML-Fragment, das JKube zum Generieren der benutzerdefinierten ConfigMap-Ressource verwendet, die die Datenbankumgebungsvariablen definiert. Sie können die erforderlichen Umgebungsvariablen in einer benutzerdefinierten ConfigMap-Ressource oder direkt in der Deploymentkonfiguration definieren.

- 5.1. Erstellen Sie die Datei `src/main/jkube/cm.yml` mit dem folgenden Inhalt:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: db-config
data:
  DATABASE_USER: todoapp
  DATABASE_PASSWORD: redhat123
  DATABASE_SVC_HOSTNAME: tododb
  DATABASE_NAME: todo
```

Alternativ ist eine Lösungs-YAML-Fragmentdatei im Verzeichnis `/home/student/DO288/solutions/todo-migrate` verfügbar. Sie können sie in das Unterverzeichnis `src/main/jkube` kopieren:

```
[student@workstation todo-api]$ cp \
~/D0288/solutions/todo-migrate/cm.yml src/main/jkube
```

6. Erstellen Sie das YAML-Fragment, das von JKube verwendet wird, um die benutzerdefinierte Deploymentkonfigurationsressource zu generieren.

- 6.1. Erstellen Sie die Datei `src/main/jkube/deployment.yml` mit dem folgenden Inhalt:

```
spec:
  template:
    spec:
      containers:
        - envFrom:
          - configMapRef:
              name: db-config
```

Der Referenzname der ConfigMap muss mit dem Namen der ConfigMap-Ressource übereinstimmen.

Alternativ ist eine Lösungs-YAML-Fragmentdatei im Verzeichnis `/home/student/D0288/solutions/todo-migrate` verfügbar. Sie können sie in das Unterverzeichnis `src/main/jkube` kopieren:

```
[student@workstation todo-api]$ cp \
~/D0288/solutions/todo-migrate/deployment.yml src/main/jkube
```

7. Wenden Sie die benutzerdefinierte ConfigMap und die Deploymentkonfigurationsressource auf das Projekt an. Verifizieren Sie, dass die Anwendung ohne Fehler bereitgestellt wird.

Verwenden Sie die externe Route, um den Zugriff auf die Anwendungsressource `todo/api/items/6` zu testen. Eine erfolgreiche Antwort gibt JSON-Daten zurück.

- 7.1. Wenden Sie die neuen OpenShift-Ressourcen auf Ihr Projekt an.

```
[student@workstation todo-api]$ mvn oc:resource oc:apply
```

- 7.2. Überprüfen Sie die vom JKube Maven-Plug-in erstellten Ressourcen.

```
[student@workstation todo-api]$ oc describe deployment/todo-api \
| grep -A1 "Environment Variables"
  Environment Variables from:
    db-config ConfigMap Optional: false
```

Der Name der ConfigMap-Ressource muss mit dem Namen der ConfigMap-Ressource übereinstimmen, die die Datenbankvariablen enthält.

```
[student@workstation todo-api]$ oc get configmap
NAME           DATA   AGE
db-config      4      5m16s
...output omitted...
```

Verifizieren Sie, ob der Anwendungs-Pod den Status Running aufweist.

```
[student@workstation todo-api]$ oc get pods
NAME                  READY   STATUS    RESTARTS   AGE
...output omitted...
todo-api-58fc84655-gs9nv   1/1     Running   0          35s
...output omitted...
```

7.3. Warten Sie, bis die Anwendung bereit ist und ausgeführt wird:

```
[student@workstation todo-api]$ oc logs -f todo-api-58fc84655-gs9nv
...output omitted...
INFO: todo-api 1.0.0-SNAPSHOT on JVM (powered by Quarkus 1.11.7.Final-redhat-00009) started in 3.183s. Listening on: http://0.0.0.0:8080
```

7.4. Verifizieren Sie, dass die Anwendung Daten von der externen Datenbank empfängt.

Führen Sie den Befehl `curl` aus, um zu testen, ob die Anwendungsressource `todo/api/items/6` JSON-Daten zurückgibt.

```
[student@workstation todo-api]$ ROUTE_URL=$(oc get route todo-api \
--template={{.spec.host}})
[student@workstation todo-api]$ curl -s ${ROUTE_URL}/todo/api/items/6 \
| jq
{
  "id": 6,
  "description": "Verify that the To Do List application works",
  "done": false
}
```

8. Nach erfolgreichen Anwendungstests committen und übertragen Sie Ihre Codeänderungen an das Remote-Repository.

```
[student@workstation todo-api]$ git add src/main/jkube/*
[student@workstation todo-api]$ git commit -m "add YAML fragments"
...output omitted...
[student@workstation todo-api]$ git push origin todo-migrate
...output omitted...
```

Bewertung

Verwenden Sie als Benutzer `student` auf dem Rechner `workstation` den Befehl `lab`, um Ihre Arbeit zu bewerten. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie den Befehl so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab todo-migrate grade
```

Beenden

Führen Sie auf der `workstation` den Befehl `lab todo-migrate finish` aus, um diese Übung zu beenden. Dieser Schritt ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken.

```
[student@workstation ~]$ lab todo-migrate finish
```

Hiermit ist die praktische Übung abgeschlossen.

Zusammenfassung

In diesem Kapitel wurden die folgenden Themen behandelt:

- Ein Servicename wird für alle Pods in einem OpenShift-Cluster zu einem lokalen DNS-Hostnamen.
- Ein externer Service wird mit dem Befehl `oc create service externalname` mit der Option `external-name` erstellt.
- Red Hat empfiehlt, dass Integritäts-Probes im Rahmen von Produktionsbereitstellungen definiert werden.
- Red Hat stellt Deploymentanwendungen in OpenShift eine Reihe an Middleware-Container-Images bereit. Dazu zählen Anwendungspakete als ausführbare JAR-Dateien.
- Das JKube Maven-Plug-in bietet Features zum Generieren von OpenShift-Ressourcen und Auslösen von OpenShift-Prozessen, beispielsweise Builds und Bereitstellungen.

Kapitel 9

Ausführliche Wiederholung: Red Hat OpenShift Development II: Containerizing Applications

Ziel

Wiederholen von Aufgaben aus *Red Hat OpenShift Development II: Containerizing Applications*

Ziele

Wiederholen von Aufgaben aus *Red Hat OpenShift Development II: Containerizing Applications*

Abschnitte

Ausführliche Wiederholung

Praktische Übung

- Erstellen und Bereitstellen von Anwendungen mit mehreren Containern in OpenShift

Ausführliche Wiederholung

Ziele

Nach Abschluss dieses Abschnitts sollten Sie die in *Red Hat OpenShift Development II: Containerizing Applications* erworbenen Kenntnisse wiederholen und auffrischen können.

Wiederholung von Red Hat OpenShift Development II: Containerizing Applications

Bevor Teilnehmer mit der ausführlichen Wiederholung für diesen Kurs beginnen, sollten sie mit den in den jeweiligen Kapiteln behandelten Themen vertraut sein.

Für zusätzliche Übungen stehen Teilnehmern auch die vorherigen Kapitel dieses Lehrbuchs zur Verfügung.

Kapitel 1, Bereitstellen und Verwalten von Anwendungen auf einem OpenShift-Cluster

Bereitstellen von Anwendungen mithilfe verschiedener Methoden für das Packen von Anwendungen auf einem OpenShift-Cluster und Verwalten der zugehörigen Ressourcen

- Beschreiben der Architektur und neuen Funktionen von OpenShift 4
- Bereitstellen einer Anwendung auf dem Cluster aus einem Dockerfile mit der CLI
- Bereitstellen einer Anwendung über ein Container-Image und Verwalten der zugehörigen Ressourcen mit der Web Console
- Bereitstellen einer Anwendung über den Quellcode und Verwalten der zugehörigen Ressourcen mithilfe der Befehlszeilenschnittstelle

Kapitel 2, Entwerfen containerisierter Anwendungen für OpenShift

Auswählen einer Methode für die Containerisierung einer Anwendung und Packen der Anwendung zur Ausführung auf einem OpenShift-Cluster

- Auswählen einer geeigneten Methode für die Containerisierung von Anwendungen
- Erstellen eines Container-Images mit zusätzlichen Anweisungen für Dockerfiles
- Auswählen einer Methode zum Einfügen von Konfigurationsdaten in eine Anwendung und Erstellen der dazu erforderlichen Ressourcen

Kapitel 3, Veröffentlichen von Enterprise Container-Images

Interagieren mit einer Unternehmens-Registry und Veröffentlichen von Container-Images in der Registry

- Verwalten von Container-Images in Registries mit Linux-Container-Tools
- Zugreifen auf die interne OpenShift-Registry mit Linux-Container-Tools

- Erstellen von Image-Streams für Container-Images in externen Registries

Kapitel 4, Verwalten von Builds auf OpenShift

Beschreiben des OpenShift-Build-Prozesses und der Trigger sowie Verwalten von Builds

- Beschreiben des OpenShift-Build-Prozesses
- Verwalten von Anwendungs-Builds mithilfe der BuildConfig-Ressource und CLI-Befehle
- Auslösen des Build-Prozesses anhand unterstützter Methoden
- Bearbeiten der Logik im Anschluss an die Erstellung mithilfe des Build-Hook „post-commit“

Kapitel 5, Anpassen von Source-to-Image-Builds

Anpassen eines vorhandenen S2I-Builder-Images und Erstellen eines neuen Builder-Images

- Beschreiben der obligatorischen und optionalen Schritte beim Source-to-Image-Build-Prozess
- Anpassen eines vorhandener S2I-Builder-Images mit Skripts
- Erstellen eines neuen S2I-Builder-Images mit S2I-Tools

Kapitel 6, Bereitstellen von Anwendungen mit mehreren Containern

Bereitstellung von Anwendungen mit mehreren Containern über Helm-Charts und Kustomize

- Beschreiben der Elemente einer OpenShift-Vorlage
- Erstellen einer Anwendung mit mehreren Containern mit Helm-Charts
- Anpassen von OpenShift-Bereitstellungen

Kapitel 7, Verwalten der Anwendungsbereitstellungen

Überwachen des Anwendungszustands und Implementieren verschiedener Deploymentmethoden für Cloud-native Anwendungen

- Implementieren von Liveness-Probes und Readiness-Probes
- Auswählen der geeigneten Deploymentstrategie für Cloud-native Anwendungen
- Verwalten der Bereitstellung einer Anwendung mithilfe von CLI-Befehlen

Die Ziele für dieses Kapitel sind in der praktischen Übung „Ausführliche Wiederholung“ nicht enthalten.

Kapitel 8, Erstellen von Anwendungen für OpenShift

Erstellen und Bereitstellen von Anwendungen auf OpenShift.

- Integrieren von containerisierten Anwendungen in nicht containerisierte Services
- Bereitstellen containerisierter Anwendungen von Drittanbietern gemäß den empfohlenen Methoden für OpenShift.
- Verwenden einer Red Hat OpenShift Application Runtime, um eine Anwendung bereitzustellen.

► Praktische Übung

Ausführliche Wiederholung

In dieser Wiederholung stellen Sie eine „To Do List“-Anwendung mit mehreren Containern in OpenShift bereitgestellt. Diese Anwendung besteht aus vier Komponenten:

- Einem MySQL-Datenbankserver (MariaDB)
- Einem auf Node.js basierenden HTTP-API-Back-End
- Einem auf React und Nginx basierenden Web-Front-End mit einer einzelnen Seite
- Einem Aufgaben-Exportservice, der den Status von Aufgaben in der Liste anzeigt

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Bereitstellen einer Anwendung mit mehreren Containern in OpenShift mithilfe einer Vielzahl von Build- und Deploymentstrategien
- Optimieren eines Containerfiles, um die Anzahl der Layer im generierten Container-Image zu reduzieren
- Erstellen und Veröffentlichen eines Container-Images in einer externen Registry
- Erstellen von Helm-Charts zum Kapseln einer wiederverwendbaren Konfiguration
- Bereitstellen einer Node.js-Anwendung anhand von Quellcode in einem Git-Repository und Weitergeben von Build-Umgebungsvariablen
- Erstellen und Nutzen von ConfigMaps zum Speichern von Anwendungskonfigurationsparametern
- Verwenden von Source-to-Image (S2I) zum Bereitstellen einer Anwendung

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen aktiven OpenShift-Cluster.
- Das MariaDB 10.3 Helm-Chart von Bitnami.
- Auf das Builder-Image für Node.js 12.
- Auf einen persönlichen GitHub-Fork und auf einen lokalen Klon des Repositorys D0288-apps, der den Quellcode der Anwendung in den Verzeichnissen `todo-frontend`, `todo-backend` und `todo-ssr` enthält

Führen Sie den folgenden Befehl auf `workstation` aus, um die Voraussetzungen zu überprüfen. Der Befehl lädt auch die Hilfs- und Lösungsdateien für die praktische Übung herunter:

```
[student@workstation ~]$ lab review-todo start
```

Spezifikationen

- Containerisieren Sie die vier Komponenten der Anwendung für ein neues OpenShift-Projekt namens \${RHT_OCP4_DEV_USER}-review-todo und stellen Sie sie bereit.

Stellen Sie sicher, dass die Bereitstellung den Empfehlungen von Red Hat in Bezug auf die Externalisierung der Konfiguration von in OpenShift bereitgestellten Anwendungen folgt.

- Back-End-Anforderungen:

Stellen Sie die Back-End- und Datenbankkomponenten mit Helm-Charts bereit.

Das Diagramm sollte das folgende Image und das folgende Tag aus Quay.io bereitstellen:
quay.io/redhattraining/todo-backend:release-46

Das Diagramm sollte immer das Image aus einer neuen Bereitstellung abrufen.

Es sollte die MariaDB 10.3-Datenbank mit der Version 9.3.11 des Bitnami-Charts als Abhängigkeit des Diagramms bereitstellen.

Bei DATABASE_USER, DATABASE_PASSWORD, DATABASE_NAME und DATABASE_SVC handelt es sich um die vom API-Pod benötigten Umgebungsvariablen.

Der Service sollte an Port 3000 gebunden sein.

Stellen Sie eine öffentliche Route für den Zugriff auf die API bereit.

- Anforderungen an das Front-End (SPA):

Rufen Sie die todo-frontend-Quellen und das Containerfile aus einem Klon Ihres persönlichen Forks des Git-Repositorys „DO288-apps“ ab.

Das bereitgestellte Containerfile generiert ein Image, das mit OCI-Container-Engines (Open Container Initiative) kompatibel ist, jedoch möglicherweise Änderungen erfordert, um den Red Hat-Empfehlungen für OpenShift zu entsprechen.

Nehmen Sie keine Änderungen an den HTML- und TypeScript-Quellen vor.

Korrigieren Sie das Container-Image, indem Sie den Benutzer nginx und Port 8080 angeben.

Minimieren Sie die Anzahl der Layer im Container-Image.

Erstellen und veröffentlichen Sie das Container-Image todo-frontend in Ihrem persönlichen Quay.io-Benutzerkonto mit quay.io/yourquayuser/front-end:latest als Namen und Tag.

Stellen Sie die „To Do List“-Benutzeroberfläche im Projekt \${RHT_OCP4_DEV_USER}-review-todo bereit.

Legen Sie die Umgebungsvariable BACKEND_HOST in der Front-End-Bereitstellung auf den Servicenamen der todo-backend-API fest.

Stellen Sie eine öffentliche Route für den Zugriff auf die Benutzeroberfläche bereit.

Testen Sie das „To Do List“-Front-End mithilfe der bereitgestellten Route.

- Anforderungen an das Front-End (statisch):

Verwenden Sie die Source-to-Image-Strategie (S2I), um die Anwendung zu erstellen und im Verzeichnis todo-ssr im Repository DO288-apps bereitzustellen.

Sie sollte den neuen Red Hat-S2I-Builder für Node.js 12 verwenden, der von OpenShift bereitgestellt wird.

Geben Sie den Anwendungsnamen `todo-ssr` für den S2I-Builder an.

Erstellen sie eine ConfigMap mit dem Namen `todo-ssr-host`, und legen Sie die Umgebungsvariable `API_HOST` so fest, dass sie auf den Service `todo-backend` auf Port 3000 verweist.

Stellen Sie eine öffentliche Route für den Zugriff auf die Benutzeroberfläche bereit.

- Tipps:

Für den Back-End-Service gibt der Endpunkt `/api/items` die aktuelle Liste der To-do-Elemente oder `[]` zurück, wenn keine Elemente vorhanden sind.

Es ist normal, dass der Anwendungs-API-Pod während der Initialisierung der Datenbank fehlschlägt und neu gestartet wird.

Wenn Sie Ihr Helm-Chart deinstallieren müssen, müssen Sie auch die Anforderung für persistente Volumes (Persistent Volume Claim, PVC) löschen, die mit dem MariaDB-Datenbankserver verknüpft ist. (Beim Löschen des Projekts wird auch die PVC gelöscht.)

Das SPA-Front-End funktioniert, wenn Sie neue To-Do-Listenelemente erstellen können, die beibehalten werden, wenn Sie die Seite aktualisieren.

Verwenden Sie den Selektor `app=todo-frontend`, um nur die UI-Ressourcen zu löschen.

Das statische Front-End funktioniert, wenn Sie eine Seite anzeigen können, auf der die von Ihnen mit der SPA-Version des Front-Ends erstellten Listenelemente aufgelistet sind.

Bewertung

Verwenden Sie als Benutzer `student` auf dem Rechner `workstation` den Befehl `lab`, um Ihre Arbeit zu bewerten. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie den Befehl so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab review-todo grade
```

Beenden

Führen Sie auf dem Rechner `workstation` als Benutzer `student` den Befehl `lab` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken.

```
[student@workstation ~]$ lab review-todo finish
```

Hiermit ist die ausführliche Wiederholung beendet.

► Lösung

Ausführliche Wiederholung

In dieser Wiederholung stellen Sie eine „To Do List“-Anwendung mit mehreren Containern in OpenShift bereitgestellt. Diese Anwendung besteht aus vier Komponenten:

- Einem MySQL-Datenbankserver (MariaDB)
- Einem auf Node.js basierenden HTTP-API-Back-End
- Einem auf React und Nginx basierenden Web-Front-End mit einer einzelnen Seite
- Einem Aufgaben-Exportservice, der den Status von Aufgaben in der Liste anzeigt

Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Bereitstellen einer Anwendung mit mehreren Containern in OpenShift mithilfe einer Vielzahl von Build- und Deploymentstrategien
- Optimieren eines Containerfiles, um die Anzahl der Layer im generierten Container-Image zu reduzieren
- Erstellen und Veröffentlichen eines Container-Images in einer externen Registry
- Erstellen von Helm-Charts zum Kapseln einer wiederverwendbaren Konfiguration
- Bereitstellen einer Node.js-Anwendung anhand von Quellcode in einem Git-Repository und Weitergeben von Build-Umgebungsvariablen
- Erstellen und Nutzen von ConfigMaps zum Speichern von Anwendungskonfigurationsparametern
- Verwenden von Source-to-Image (S2I) zum Bereitstellen einer Anwendung

Bevor Sie Beginnen

Sie müssen auf Folgendes zugreifen können, um diese Übung durchführen zu können:

- Auf einen aktiven OpenShift-Cluster.
- Das MariaDB 10.3 Helm-Chart von Bitnami.
- Auf das Builder-Image für Node.js 12.
- Auf einen persönlichen GitHub-Fork und auf einen lokalen Klon des Repositorys D0288-apps, der den Quellcode der Anwendung in den Verzeichnissen `todo-frontend`, `todo-backend` und `todo-ssr` enthält

Führen Sie den folgenden Befehl auf `workstation` aus, um die Voraussetzungen zu überprüfen. Der Befehl lädt auch die Hilfs- und Lösungsdateien für die praktische Übung herunter:

```
[student@workstation ~]$ lab review-todo start
```

Spezifikationen

- Containerisieren Sie die vier Komponenten der Anwendung für ein neues OpenShift-Projekt namens \${RHT_OCP4_DEV_USER}-review-todo und stellen Sie sie bereit.

Stellen Sie sicher, dass die Bereitstellung den Empfehlungen von Red Hat in Bezug auf die Externalisierung der Konfiguration von in OpenShift bereitgestellten Anwendungen folgt.

- Back-End-Anforderungen:

Stellen Sie die Back-End- und Datenbankkomponenten mit Helm-Charts bereit.

Das Diagramm sollte das folgende Image und das folgende Tag aus Quay.io bereitstellen:
quay.io/redhattraining/todo-backend:release-46

Das Diagramm sollte immer das Image aus einer neuen Bereitstellung abrufen.

Es sollte die MariaDB 10.3-Datenbank mit der Version 9.3.11 des Bitnami-Charts als Abhängigkeit des Diagramms bereitstellen.

Bei DATABASE_USER, DATABASE_PASSWORD, DATABASE_NAME und DATABASE_SVC handelt es sich um die vom API-Pod benötigten Umgebungsvariablen.

Der Service sollte an Port 3000 gebunden sein.

Stellen Sie eine öffentliche Route für den Zugriff auf die API bereit.

- Anforderungen an das Front-End (SPA):

Rufen Sie die todo-frontend-Quellen und das Containerfile aus einem Klon Ihres persönlichen Forks des Git-Repositorys „DO288-apps“ ab.

Das bereitgestellte Containerfile generiert ein Image, das mit OCI-Container-Engines (Open Container Initiative) kompatibel ist, jedoch möglicherweise Änderungen erfordert, um den Red Hat-Empfehlungen für OpenShift zu entsprechen.

Nehmen Sie keine Änderungen an den HTML- und TypeScript-Quellen vor.

Korrigieren Sie das Container-Image, indem Sie den Benutzer nginx und Port 8080 angeben.

Minimieren Sie die Anzahl der Layer im Container-Image.

Erstellen und veröffentlichen Sie das Container-Image todo-frontend in Ihrem persönlichen Quay.io-Benutzerkonto mit quay.io/yourquayuser/front-end:latest als Namen und Tag.

Stellen Sie die „To Do List“-Benutzeroberfläche im Projekt \${RHT_OCP4_DEV_USER}-review-todo bereit.

Legen Sie die UmgebungsvARIABLE BACKEND_HOST in der Front-End-Bereitstellung auf den Servicenamen der todo-backend-API fest.

Stellen Sie eine öffentliche Route für den Zugriff auf die Benutzeroberfläche bereit.

Testen Sie das „To Do List“-Front-End mithilfe der bereitgestellten Route.

- Anforderungen an das Front-End (statisch):

Verwenden Sie die Source-to-Image-Strategie (S2I), um die Anwendung zu erstellen und im Verzeichnis todo-ssr im Repository DO288-apps bereitzustellen.

Sie sollte den neuen Red Hat-S2I-Builder für Node.js 12 verwenden, der von OpenShift bereitgestellt wird.

Geben Sie den Anwendungsnamen `todo-ssr` für den S2I-Builder an.

Erstellen sie eine ConfigMap mit dem Namen `todo-ssr-host`, und legen Sie die Umgebungsvariable `API_HOST` so fest, dass sie auf den Service `todo-backend` auf Port 3000 verweist.

Stellen Sie eine öffentliche Route für den Zugriff auf die Benutzeroberfläche bereit.

- Tipps:

Für den Back-End-Service gibt der Endpunkt `/api/items` die aktuelle Liste der To-do-Elemente oder `[]` zurück, wenn keine Elemente vorhanden sind.

Es ist normal, dass der Anwendungs-API-Pod während der Initialisierung der Datenbank fehlschlägt und neu gestartet wird.

Wenn Sie Ihr Helm-Chart deinstallieren müssen, müssen Sie auch die Anforderung für persistente Volumes (Persistent Volume Claim, PVC) löschen, die mit dem MariaDB-Datenbankserver verknüpft ist. (Beim Löschen des Projekts wird auch die PVC gelöscht.)

Das SPA-Front-End funktioniert, wenn Sie neue To-Do-Listenelemente erstellen können, die beibehalten werden, wenn Sie die Seite aktualisieren.

Verwenden Sie den Selektor `app=todo-frontend`, um *nur* die UI-Ressourcen zu löschen.

Das statische Front-End funktioniert, wenn Sie eine Seite anzeigen können, auf der die von Ihnen mit der SPA-Version des Front-Ends erstellten Listenelemente aufgelistet sind.

1. Erstellen Sie in einem neuen OpenShift-Projekt namens `${RHT_OCP4_DEV_USER} -review-todo` ein Helm-Chart, das das Node.js-Back-End bereitstellt. Das neue Diagramm sollte vom Bitnami MariaDB-Helm-Chart abhängig sein. Erstellen Sie eine neue Route zur API, damit sie öffentlich verfügbar ist.

- 1.1. Laden Sie die Konfiguration Ihrer Kursumgebung.

Führen Sie den folgenden Befehl aus, um die in der ersten angeleiteten Übung erstellten Umgebungsvariablen zu laden:

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

- 1.2. Melden Sie sich bei OpenShift mit Ihrem Entwicklerbenutzerkonto an:

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} \
-p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful
...output omitted...
```

- 1.3. Erstellen Sie ein neues OpenShift-Projekt mit dem Namen `${RHT_OCP4_DEV_USER} -review-todo`.

```
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-review-todo
Now using project "youruser-review-todo" on server ...output omitted...
...output omitted...
```

- 1.4. Initialisieren Sie in Ihrem Übungsverzeichnis ein neues Helm-Chart mit dem Namen `todo-list`, und wechseln Sie in das neu erstellte Verzeichnis.

```
[student@workstation ~]$ cd ~/D0288/labs/review-todo
[student@workstation review-todo]$ helm create todo-list
Creating todo-list
[student@workstation review-todo]$ cd todo-list
[student@workstation todo-list]$
```



Anmerkung

Eine fertige Version des Helm-Charts wird in `~/D0288/solutions/review-todo/todo-list` bereitgestellt. Wenn Sie sich nicht sicher sind, können Sie auf die angegebene Lösung verweisen oder sie kopieren.

- 1.5. Hängen Sie Folgendes an den Inhalt der Datei `Chart.yaml` an, die das MariaDB-Diagramm als Abhängigkeit deklariert.

```
dependencies:
- name: mariadb
  version: 9.3.11
  repository: https://charts.bitnami.com/bitnami
```

- 1.6. Rufen Sie die Abhängigkeiten ab.

```
[student@workstation todo-list]$ helm dependency update
Getting updates for unmanaged Helm repositories...
...Successfully got an update from the "https://charts.bitnami.com/bitnami" chart
repository
Saving 1 charts
Downloading mariadb from repo https://charts.bitnami.com/bitnami
Deleting outdated charts
```

- 1.7. Aktualisieren Sie in der Datei `values.yaml` die Werte im Abschnitt `image`.

```
image:
  repository: quay.io/redhattraining/todo-backend
  pullPolicy: Always
  # Overrides the image tag whose default is the chart appVersion.
  tag: "release-46"
```

- 1.8. Hängen Sie die folgenden Abschnitte an das Ende von `values.yaml` an:

```
mariadb:
  auth:
    username: todouser
```

```
password: todopwd
database: tododb
primary:
  podSecurityContext:
    enabled: false
  containerSecurityContext:
    enabled: false

env:
  - name: DATABASE_NAME
    value: tododb
  - name: DATABASE_USER
    value: todouser
  - name: DATABASE_PASSWORD
    value: todopwd
  - name: DATABASE_SVC
    value: todo-list-mariadb
```

- 1.9. Aktualisieren Sie ebenfalls in `values.yaml` den Wert `port` unter `service` auf 3000.

```
service:
  type: ClusterIP
  port: 3000
```

- 1.10. Aktualisieren Sie `templates/deployment.yaml`, um im Abschnitt `containers` einen `env`-Abschnitt hinzuzufügen, und legen Sie `containerPort` auf 3000 fest. Stellen Sie sicher, dass die Einrückung dem folgenden Beispiel entspricht.

```
apiVersion: apps/v1
...output omitted...
spec:
  ...output omitted...
  template:
    spec:
      ...output omitted...
      containers:
        - name: {{ .Chart.Name }}
          securityContext:
            {{- toYaml .Values.securityContext | nindent 12 }}
            image: "{{ .Values.image.repository }}:{{ .Values.image.tag | default .Chart.AppVersion }}"
            imagePullPolicy: {{ .Values.image.pullPolicy }}
          env:
            {{- range .Values.env }}
            - name: {{ .name }}
              value: {{ .value }}
            {{- end }}
          ports:
            - name: http
              containerPort: 3000
              protocol: TCP
```

- 1.11. Installieren Sie das Diagramm, und geben Sie den Namen `todo-list` an:

```
[student@workstation todo-list]$ helm install todo-list .
NAME: todo-list
LAST DEPLOYED: ...output omitted...
NAMESPACE: youruser-review-todo
STATUS: deployed
REVISION: 1
...output omitted...
```

Dadurch werden alle Ressourcen erstellt, die vom Helm-Chart im aktuell ausgewählten Projekt definiert sind.

- 1.12. Vergewissern Sie sich, dass die Anwendung erfolgreich gestartet wurde:

```
[student@workstation todo-list]$ oc get pods
[student@workstation review-todo]$ oc get pods
NAME                  READY   STATUS    RESTARTS   AGE
todo-list-7b6dbb8ccb-bqvz   1/1     Running   2          5m57s
todo-list-mariadb-0         1/1     Running   0          5m57s
```



Anmerkung

Wenn Ihr API-Pod weiterhin abstürzt und neu startet, müssen Sie Ihr Helm-Chart deinstallieren, korrigieren und es erneut versuchen.

- 1.13. Erstellen Sie eine neue Route zur API.

```
[student@workstation todo-list]$ oc expose svc/todo-list
route.route.openshift.io/todo-list exposed
```

- 1.14. Rufen Sie die öffentliche URL zum Service ab.

```
[student@workstation todo-list]$ export URL_TO_APPLICATION=$(oc get \
route/todo-list -o jsonpath='{.spec.host}')
...output omitted...
```

- 1.15. Verwenden Sie die URL, um zu überprüfen, ob die API gestartet wurde, indem Sie eine Verbindung zur API herstellen.

```
[student@workstation todo-list]$ curl ${URL_TO_APPLICATION}
OK
```

- 1.16. Verwenden Sie die URL, um zu überprüfen, dass der Service mit der Datenbank verbunden ist.

```
[student@workstation todo-list]$ curl ${URL_TO_APPLICATION}/api/items
[]
```

2. Erstellen Sie die React-Benutzeroberfläche in OpenShift mithilfe von `new-app`, indem Sie ein Image an Quay.io übertragen. Korrigieren und erstellen Sie das Image `todo-frontend`

so, dass es den Benutzer `nginx` verwendet, Port 8080 bereitstellt und die Anzahl der Layer minimiert. Erstellen Sie eine neue Route zur Benutzeroberfläche, damit sie öffentlich verfügbar ist.

- 2.1. Öffnen Sie auf Ihrem Fork von D0288-apps die Datei `todo-frontend/Containerfile`, und kombinieren Sie die separaten RUN-Befehle in einer einzelnen Anweisung.

```
RUN cd /tmp/todo-frontend && \
  npm install && \
  npm run build
```



Anmerkung

Eine feste Version des Containerfiles wird in `~/D0288/solutions/review-todo/Containerfile-frontend-solution` bereitgestellt. Wenn Sie sich nicht sicher sind, können Sie auf die angegebene Lösung verweisen oder sie kopieren.

- 2.2. Fügen Sie ebenfalls in der Datei `todo-frontend/Containerfile` die Befehle `EXPOSE` und `USER` hinzu:

```
COPY --from=appbuild /tmp/todo-frontend/build /usr/share/nginx/html

EXPOSE 8080

USER nginx

CMD nginx -g "daemon off;"
```

- 2.3. Erstellen Sie das Image lokal mit Podman.

```
[student@workstation todo-frontend]$ cd ~/D0288-apps/todo-frontend/
[student@workstation todo-frontend]$ podman build . \
-t quay.io/${RHT_OCP4_QUAY_USER}/todo-frontend:latest
STEP 1: FROM registry.access.redhat.com/ubi8/nodejs-14 AS appbuild
Getting image source signatures
...output omitted...
STEP 18: COMMIT quay.io/youruser/todo-frontend:latest
...output omitted...
```

Beachten Sie, dass dieser Schritt möglicherweise einige Minuten in Anspruch nimmt.

- 2.4. Navigieren Sie in einem Browser zu Quay.io, und erstellen Sie ein neues leeres Repository mit dem Namen `todo-frontend`. Legen Sie unter dem Feld „name“ das Repository als öffentlich fest. Andernfalls kann OpenShift nicht darauf zugreifen.
- 2.5. Melden Sie sich mit Podman bei Ihrem persönlichen Quay.io-Benutzerkonto an.

```
[student@workstation ~]$ podman login -u ${RHT_OCP4_QUAY_USER} quay.io
Password:
Login Succeeded!
```

- 2.6. Übertragen Sie das Image mit Podman an Quay.io.

```
[student@workstation ~]$ podman push quay.io/${RHT_OCP4_QUAY_USER}/todo-frontend
Getting image source signatures
...output omitted...
Writing manifest to image destination
Storing signatures
```

Beachten Sie, dass Sie sich bei Quay.io authentifizieren müssen.

- 2.7. Stellen Sie das Image mithilfe des Image-Streams über `oc new-app` bereit.

```
[student@workstation ~]$ oc new-app quay.io/${RHT_OCP4_QUAY_USER}/todo-frontend
--> Found container image ...output omitted...
...output omitted...
--> Creating resources ...
  imagestream.image.openshift.io "todo-frontend" created
  deployment.apps "todo-frontend" created
  service "todo-frontend" created
--> Success
...output omitted...
```

- 2.8. Vergewissern Sie sich, dass die Anwendung erfolgreich gestartet wurde:

```
[student@workstation ~]$ oc get pods
NAME                      READY   STATUS    RESTARTS   AGE
...output omitted...
todo-frontend-7b4d77b4f8-lsrpm   1/1     Running   0          1m20s
```

- 2.9. Erstellen Sie eine Route zum Service mit dem Befehl „expose“:

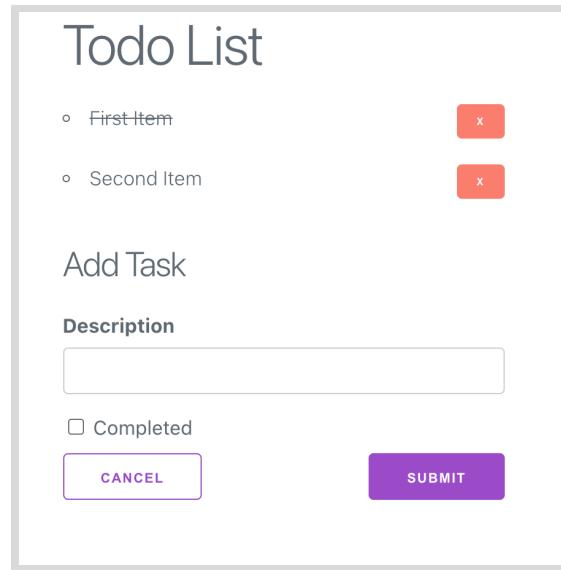
```
[student@workstation ~]$ oc expose svc/todo-frontend
route.route.openshift.io/todo-frontend exposed
```

- 2.10. Rufen Sie die öffentliche URL ab, indem Sie die Route untersuchen.

```
[student@workstation ~]$ oc get route todo-frontend -o jsonpath='{.spec.host}'
```

- 2.11. Überprüfen Sie, ob die Benutzeroberfläche funktioniert, indem Sie die abgerufene URL in einem Browser öffnen. Die Benutzeroberfläche sollte eine „To Do List“-SPA-Anwendung enthalten.

- 2.12. Überprüfen Sie, ob Sie „To Do List“-Elemente erstellen können, die beibehalten werden, wenn Sie die Seite aktualisieren.



3. Stellen Sie eine serverseitige gerenderte Node.js-Anwendung namens todo-ssr mit S2I bereit. Der Quellcode ist im Repository „DO288-apps“ im Verzeichnis todo-ssr verfügbar. Stellen Sie sicher, dass der Service unter Verwendung von Port 3000 ausgeführt wird. Erstellen Sie eine neue Route zur Benutzeroberfläche, damit sie öffentlich verfügbar ist.

3.1. Initialisieren Sie mit new-app die Anwendung todo-ssr mit S2I.

```
[student@workstation ~]$ oc new-app \
https://github.com/RedHatTraining/DO288-apps \
--name todo-ssr --context-dir=todo-ssr --build-env \
npm_config_registry="${RHT_OCP4_NEXUS_SERVER}/repository/nodejs"
--> Found image 9350f28 (5 months old) in image stream "openshift/nodejs" under
tag "12-ubi8" for "nodejs"
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "todo-ssr" created
buildconfig.build.openshift.io "todo-ssr" created
deployment.apps "todo-ssr" created
service "todo-ssr" created
--> Success
...output omitted...
```

3.2. Erstellen Sie eine ConfigMap mit dem Namen todo-ssr-host, die die Umgebungsvariable API_HOST enthält.

```
[student@workstation ~]$ oc create configmap todo-ssr-host \
--from-literal API_HOST="http://todo-list:3000"
configmap/todo-ssr-host created
```

3.3. Verbinden Sie die ConfigMap mit der Bereitstellung todo-ssr.

```
[student@workstation ~]$ oc set env deployment/todo-ssr \
--from cm/todo-ssr-host
deployment.apps/todo-ssr updated
```

Beachten Sie, dass es nach Anhängen der ConfigMap möglicherweise einige Minuten dauert, bis die Anwendung neu gestartet wird.

- 3.4. Vergewissern Sie sich, dass die Anwendung erfolgreich gestartet wurde:

```
[student@workstation ~]$ oc get pods
NAME                  READY   STATUS    RESTARTS   AGE
...output omitted...
todo-ssr-1-build      0/1     Completed  0          6m30s
todo-ssr-64bc5f8987-7654f 1/1     Running   0          1m12s
```

- 3.5. Erstellen Sie eine Route zum Service mit dem Befehl „expose“.

```
[student@workstation ~]$ oc expose svc/todo-ssr
route.route.openshift.io/todo-ssr exposed
```

- 3.6. Rufen Sie die öffentliche URL ab, indem Sie die Route untersuchen.

```
[student@workstation ~]$ oc get route todo-ssr -o jsonpath='{.spec.host}'
...output omitted...
```

- 3.7. Überprüfen Sie, ob die statische Benutzeroberfläche funktioniert, indem Sie die abgerufene URL in einem Browser öffnen. Die Benutzeroberfläche sollte eine statische Seite mit den „To Do Liste“-Elementen enthalten, die Sie in der SPA-Version der Benutzeroberfläche erstellt haben.

Todo List

Note that this version is non-interactive

- foo
- baz

Bewertung

Verwenden Sie als Benutzer student auf dem Rechner workstation den Befehl lab, um Ihre Arbeit zu bewerten. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie den Befehl so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab review-todo grade
```

Beenden

Führen Sie auf dem Rechner `workstation` als Benutzer `student` den Befehl `lab` aus, um diese Übung zu beenden. Dies ist wichtig, um sicherzustellen, dass Ressourcen aus vorherigen Übungen sich nicht auf zukünftige Übungen auswirken.

```
[student@workstation ~]$ lab review-todo finish
```

Hiermit ist die ausführliche Wiederholung beendet.

Anhang A

Erstellen eines GitHub-Benutzerkontos

Ziel

Beschreiben, wie ein GitHub-Benutzerkonto für praktische Übungen im Kurs erstellt wird

Erstellen eines GitHub-Benutzerkontos

Ziele

Nach Abschluss dieses Abschnitts sollten Sie in der Lage sein, ein GitHub-Benutzerkonto sowie öffentliche Git-Repositorys zu erstellen.

Erstellen eines GitHub-Benutzerkontos

Sie benötigen ein GitHub-Benutzerkonto, um ein oder mehrere *öffentliche* Git-Repositorys für die Labs in diesem Kurs zu erstellen. Falls Sie bereits ein GitHub-Benutzerkonto besitzen, können Sie die in diesem Anhang aufgelisteten Schritte überspringen.



Wichtig

Stellen Sie sicher, dass Sie für die Labs in diesem Kurs nur *öffentliche* Git-Repositorys erstellen. Die Übungsskripts und -anweisungen zur Bewertung der Labs erfordern authentifizierte Zugriff, um die Repositorys zu klonen.

Um ein neues GitHub-Benutzerkonto zu erstellen, navigieren Sie zu <https://github.com>, klicken Sie auf **Sign up**, und folgen Sie den Eingabeaufforderungen. Sie erhalten eine E-Mail mit Anweisungen zur Aktivierung Ihres GitHub-Benutzerkontos. Überprüfen Sie Ihre E-Mail-Adresse und melden Sie sich dann auf der GitHub-Website mit dem Benutzernamen und dem Passwort an, die Sie bei der Kontoerstellung angegeben haben.

Erstellen von GitHub-Repositories

Erstellen Sie nach der Authentifizierung neue Git-Repositorys. Klicken Sie dazu im Bereich **Repositories** auf der linken Seite der GitHub-Startseite auf **New**.

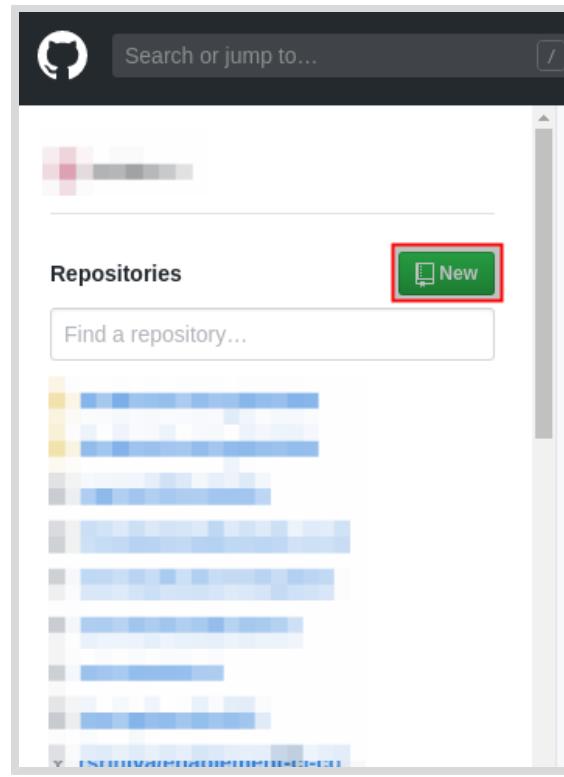


Abbildung A.1: Erstellen eines neuen Git-Repositorys

Alternativ können Sie auf das Pluszeichen (+) in der rechten oberen Ecke (rechts neben dem Glockensymbol) und dann auf New repository klicken.

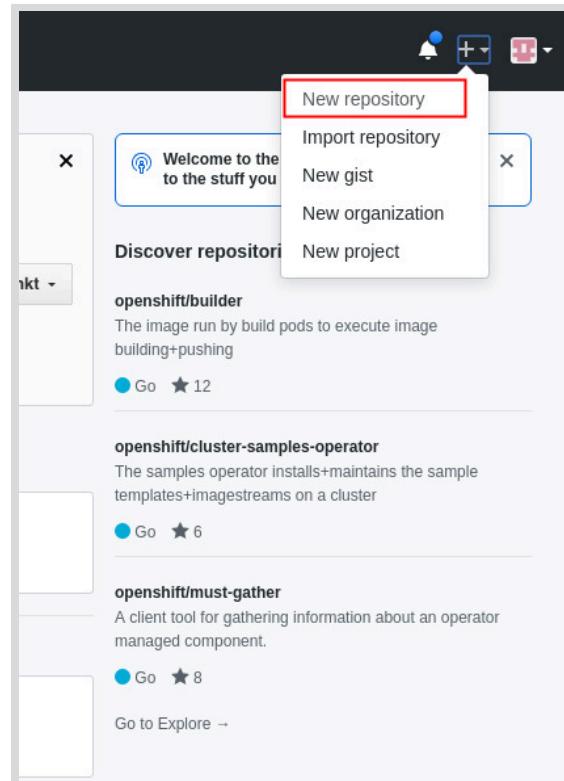


Abbildung A.2: Erstellen eines neuen Git-Repositorys

Forken von GitHub-Repositories

Um ein Repository in GitHub zu forken, navigieren Sie zum Repository, und klicken Sie oben rechts auf **Fork**. Klicken Sie auf **Fork** und nicht auf die **Zahl** neben der Schaltfläche.

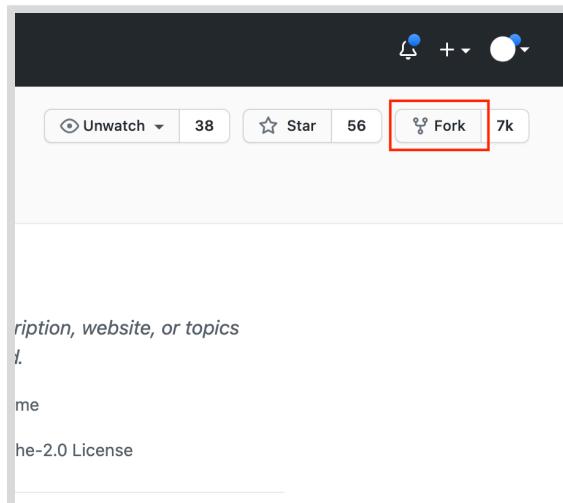


Abbildung A.3: Forken eines Git-Repositorys

Wählen Sie Ihren Benutzernamen als Ziel aus, und warten Sie, bis der Prozess abgeschlossen ist.

Wenn Sie das Repository auf Ihre Workstation klonen, verwenden Sie die URL für Ihren Fork. Diesen finden Sie, wenn Sie auf **Code** klicken.

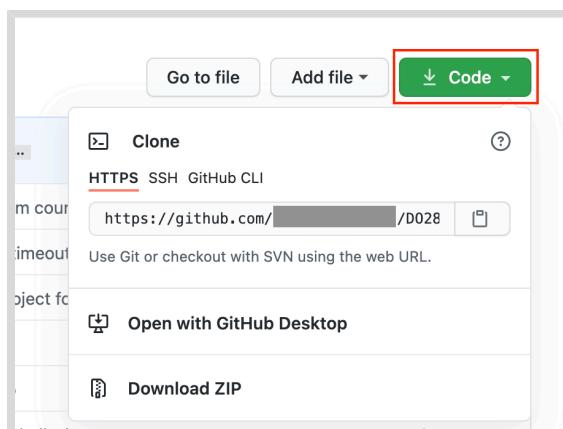


Abbildung A.4: Klonen eines geforkten Repositorys

Aktualisieren Ihres Fork

Wenn die Repository RedHatTraining-Kopie des Repositorys aktualisiert wurde, müssen Sie Ihren Fork aktualisieren. Um Ihren Fork zu aktualisieren, fügen Sie Ihrer lokalen Kopie des Repositorys eine Remote-Kopie hinzu, rufen Sie die Änderungen ab, und übertragen Sie dann die Änderungen an Ihren Fork.

Anhang A | Erstellen eines GitHub-Benutzerkontos

```
[student@workstation D0288-apps]$ git remote add upstream \
https://github.com/RedHatTraining/D0288-apps.git
[student@workstation D0288-apps]$ git pull upstream main
...output omitted...
[student@workstation D0288-apps]$ git push origin main
...output omitted...
```

Erstellen eines persönlichen GitHub-Zugriffstokens

Die passwortbasierte Authentifizierung wird von GitHub nicht mehr unterstützt. Dies bedeutet, dass Sie ein persönliches Zugriffstoken verwenden müssen, um Git-Befehle auszuführen, für die eine Authentifizierung erforderlich ist, z. B. Klonen privater Repositorys oder Pushen von Änderungen an GitHub.

Verwenden Sie die Anleitung Erstellen eines persönlichen Zugriffstokens [<https://docs.github.com/en/github/authenticating-to-github/keeping-your-account-and-data-secure/creating-a-personal-access-token>] zum Erstellen eines persönlichen Zugriffstokens, falls Sie nicht bereits über eins verfügen.

Bei einigen Übungen in diesem Kurs müssen Sie sich bei GitHub authentifizieren. Wenn Sie nach Ihrem Passwort gefragt werden, geben Sie Ihr persönliches Zugriffstoken ein.



Literaturhinweise

Registrieren eines neuen GitHub-Benutzerkontos

<https://help.github.com/en/articles/signing-up-for-a-new-github-account>

Erstellen eines persönlichen Zugriffstokens

<https://docs.github.com/en/github/authenticating-to-github/keeping-your-account-and-data-secure/creating-a-personal-access-token>

Anhang B

Erstellen eines Quay-Benutzerkontos

Ziel

Beschreiben, wie ein Quay-Benutzerkonto für praktische Übungen im Kurs erstellt wird

Erstellen eines Quay-Benutzerkontos

Ziele

In diesem Abschnitt lernen Sie, wie Sie ein Quay-Benutzerkonto erstellen und ein verschlüsseltes Passwort für die Übungen im Kurs verwenden, für die Quay.io-Zugriff erforderlich ist.

Erstellen eines Quay-Kontos

Sie benötigen ein Quay-Konto, um ein oder mehrere Container-Image-Repositorys für die Übungen in diesem Kurs zu erstellen. Wenn Sie bereits ein Quay-Benutzerkonto besitzen, können Sie die Schritte zum Erstellen eines neuen Benutzerkontos überspringen, die in diesem Anhang aufgeführt sind.

Führen Sie die folgenden Schritte aus, um ein neues Quay-Konto zu erstellen:

1. Navigieren Sie mit einem Webbrowser zu <https://quay.io>.
2. Klicken Sie in der rechten oberen Ecke (neben der Suchleiste) auf **Sign in**.
3. Melden Sie sich auf der Seite **Sign in** mit Ihren Red Hat-Anmelddaten an. Melden Sie sich mit Red Hat-Anmelddaten an. image::images/appendix/quay-signup.png[align="center",width="50%"]



Warnung

Nach dem 31. Juli 2021 ist die Red Hat Anmeldung die einzige verfügbare Webanmeldeooption.

4. Melden Sie sich mit Ihren Red Hat-SSO-Anmelddaten bei Quay.io an. Wenn Sie kein Red Hat-SSO-Benutzerkonto besitzen, verwenden Sie den Link [create one now](#) unten im Anmeldeialog, um zunächst ein kostenloses Red Hat-SSO-Benutzerkonto zu erstellen.

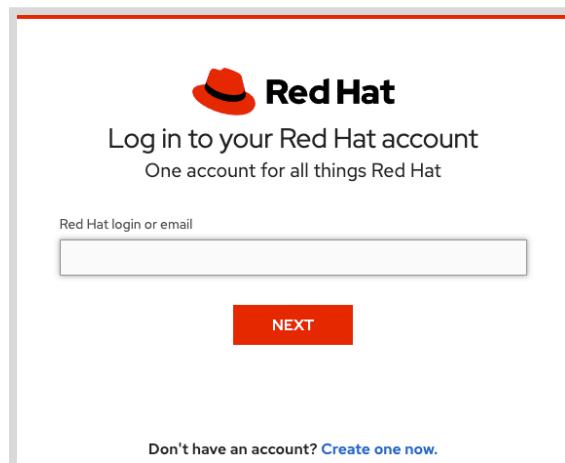


Abbildung B.1: Erstellen eines neuen Kontos

Erstellen eines Image-Repositorys

In den praktischen Übungen werden Image-Repositorys über die Befehlszeilentools erstellt. Sie können sie jedoch auch in der Weboberfläche erstellen.

1. Nachdem Sie sich bei Quay.io angemeldet haben, können Sie neue Image-Repositorys erstellen, indem Sie auf der Seite „Repositories“ auf **Create New Repository** klicken.
2. Alternativ können Sie auf das Pluszeichen (+) in der rechten oberen Ecke (links neben dem Glockensymbol) und dann auf **Create New Repository** klicken.

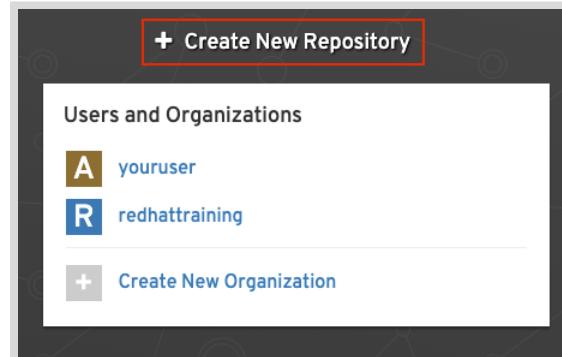


Abbildung B.2: Erstellen eines neuen Image-Repositorys

Veröffentlichen eines Repositorys

Standardmäßig werden neue Repositorys *privat* erstellt. Dies ist für die praktischen Übungen, in denen Login-Secrets verwendet werden, ausreichend, für einige Übungen sind jedoch *öffentliche* Repositorys erforderlich. Private Repositorys weisen neben dem Repository-Namen ein kleines Schlosssymbol auf.



Abbildung B.3: Privates Repository

Mit den folgenden Schritten machen Sie ein Repository öffentlich.

1. Wählen Sie den Link für Ihr Repository aus.
2. Klicken Sie im Menü auf der linken Seite auf das Symbol *Settings*.

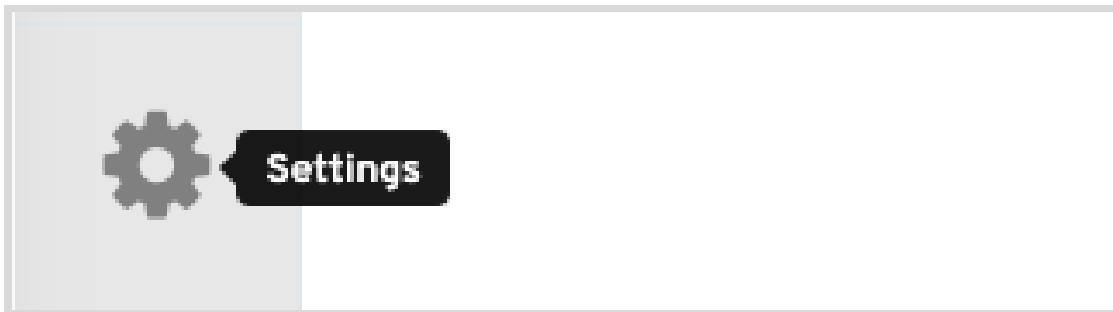


Abbildung B.4: Quay-Repository-Einstellungen

3. Klicken Sie auf der Seite für Einstellungen im Abschnitt **Repository Visibility** auf **Make Public**.

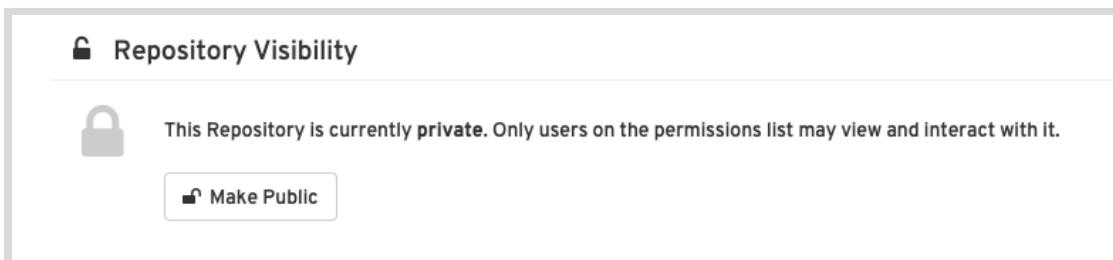


Abbildung B.5: Repository öffentlich machen

Arbeiten mit CLI-Tools

Wenn Sie Ihr Benutzerkonto mit Red Hat-SSO erstellt haben, müssen Sie ein Benutzerkontopasswort festlegen, um CLI-Tools wie Podman zu verwenden.

1. Klicken Sie oben rechts auf Ihren Namen.
2. Klicken Sie auf **Account Settings**.
3. Klicken Sie auf **Change password**.

In der Podman-CLI werden in der Befehlszeile eingegebene Passwörter in Klartext gespeichert. Es wird daher dringend empfohlen, eine verschlüsselte Version Ihres Passworts für die Verwendung mit `podman login` zu generieren.

4. Klicken Sie oben auf der Seite **Account Settings** auf **Generate Encrypted Password**.
5. Geben Sie bei Anzeige der Eingabeaufforderung Ihr Benutzerkontopasswort ein, und generieren Sie das verschlüsselte Passwort.

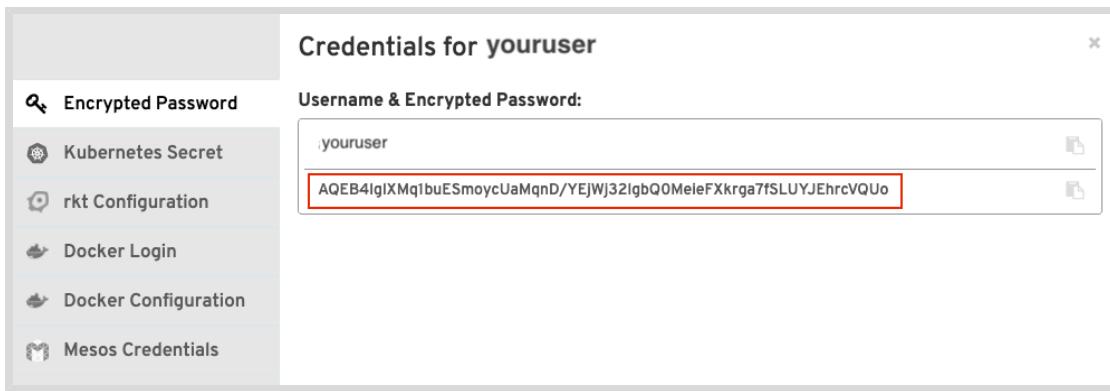


Abbildung B.6: Verschlüsseltes Quay-Passwort

6. Verwenden Sie in den praktischen Übungen das verschlüsselte Passwort, wenn Sie nach einem Quay.io-Passwort gefragt werden.

```
[student@workstation]$ podman login -u ${RHT_OCP4_QUAY_USER} quay.io
Password: <use your encrypted password here>
Login Succeeded!
```



Literaturhinweise

Erste Schritte mit Quay.io

<https://docs.quay.io/solution/getting-started.html>

Anhang C

Nützliche Git-Befehle

Ziel

Beschreiben der nützlichen Git-Befehle, die für die praktischen Übungen in diesem Kurs verwendet werden

Git-Befehle

Ziele

Nach Abschluss dieses Abschnitts sollten Sie in der Lage sein, die Übungen in diesem Kurs neu zu starten und zu wiederholen. Sie sollten ferner in der Lage sein, von einer unvollständigen Übung zu einer anderen zu wechseln, und später die vorherige Übung da fortzusetzen, wo Sie aufgehört haben.

Arbeiten mit Git-Banches

In diesem Kurs wird ein auf GitHub gehostetes Git-Repository verwendet, um den Anwendungsquellcode des Kurses zu speichern. Zu Beginn des Kurses erstellen Sie einen eigenen Fork dieses Repositorys, das auch auf GitHub gehostet wird. Sie arbeiten in diesem Kurs mit einer lokalen Kopie Ihres Fork, den Sie auf die `workstation`-VM klonen.

Der Begriff `origin` bezieht sich auf das Remote-Repository, von dem ein lokales Repository geklont wird.

Für die Übungen im Kurs verwenden Sie jeweils einen separaten Git-Branch. Alle Änderungen, die Sie am Quellcode vornehmen, erfolgen in einem neuen Branch, den Sie nur für diese Übung erstellen. Committen Sie niemals Änderungen am `main`-Branch.

Nachfolgend finden Sie eine Liste von Szenarien und die entsprechenden Befehle, die Sie verwenden können.

Vollständiges Beenden und Neustarten einer Übung

Führen Sie die folgenden Schritte aus, um eine bereits abgeschlossene Übung von Anfang an zu wiederholen:

- Übergeben und übertragen Sie im Rahmen der Übung alle Änderungen in Ihren lokalen Branch. Schließen Sie die Übung ab, indem Sie den Unterbefehl `finish` ausführen, um alle Ressourcen zu bereinigen:

```
[student@workstation ~]$ lab your-exercise finish
```

- Wechseln Sie zu Ihrem lokalen Klon des D0288-apps-Repository, und wechseln Sie in den `master`-Branch:

```
[student@workstation ~]$ cd ~/D0288-apps
[student@workstation D0288-apps]$ git checkout main
```

- Löschen Sie Ihren lokalen Branch:

```
[student@workstation D0288-apps]$ git branch -d your-branch
```

- Löschen Sie den Remote-Branch in Ihrem persönlichen GitHub-Benutzerkonto:

```
[student@workstation D0288-apps]$ git push origin --delete your-branch
```

5. Verwerfen Sie alle ausstehenden Änderungen mit `git reset --hard`:

```
[student@workstation D0288-apps]$ git reset --hard
```

6. Wenn Sie Änderungen haben, die Sie später verwenden möchten, die Sie aber müssen vorübergehend verwerfen müssen, verwenden Sie `git stash -u`:

```
[student@workstation D0288-apps]$ git stash -u
```

7. Führen Sie den Unterbefehl `start` aus, um die Übung neu zu starten:

```
[student@workstation D0288-apps]$ cd ~
[student@workstation ~]$ lab your-exercise start
```

Wechseln zu einer anderen Übung aus einer unvollständigen Übung

Es kann vorkommen, dass Sie einige Schritte in einer Übung teilweise abgeschlossen haben, aber Sie möchten zu einer anderen Übung wechseln und die aktuelle Übung zu einem späteren Zeitpunkt erneut aufrufen.

Vermeiden Sie es, zu viele Übungen unabgeschlossen zu belassen, um sie später erneut zu bearbeiten. In diesen Übungen werden gemeinsam genutzte Ressourcen reserviert, und Sie könnten Ihr Kontingent beim Cloud-Provider und im RHOC-Cluster ausschöpfen. Wenn Sie der Meinung sind, dass es möglicherweise eine Weile dauert, bis Sie zur aktuellen Übung zurückkehren können, sollten Sie diese Aufgabe abbrechen und später neu starten.

Wenn Sie die aktuelle Übung unterbrechen und an der nächsten Übung arbeiten möchten, führen Sie die folgenden Schritte aus:

1. Überprüfen Sie, welche Dateien Sie geändert haben, um zu sehen, was zu Ihrem Commit hinzugefügt werden soll.
2. Fügen Sie Dateien mit zu speichernden Änderungen hinzu.
3. Übergeben Sie die Änderungen in Ihr lokales Repository, und übertragen Sie diese an Ihr persönliches GitHub-Benutzerkonto. Möglicherweise möchten Sie den Schritt aufzeichnen, in dem Sie die Übung angehalten haben:

```
[student@workstation ~]$ cd ~/D0288-apps
[student@workstation D0288-apps]$ git status
...output omitted...
[student@workstation D0288-apps]$ git add some/file/you/changed
[student@workstation D0288-apps]$ git commit -m 'Paused at step X.Y'
[student@workstation D0288-apps]$ git push origin branch-name
```

4. Führen Sie den Befehl `finish` der ursprünglichen Übung nicht aus. Dies ist wichtig, um Ihre bestehenden OpenShift-Projekte unverändert zu belassen, sodass Sie später fortfahren können.
5. Starten Sie die nächste Übung, indem Sie den Unterbefehl `start` ausführen:

Anhang C | Nützliche Git-Befehle

```
[student@workstation ~]$ lab your-exercise start
```

6. Die nächste Übung wechselt zum `main`-Branch und erstellt optional einen neuen Branch für Änderungen. Das bedeutet, dass die Änderungen, die an der ursprünglichen Übung im ursprünglichen Branch vorgenommen wurden, unverändert bleiben.

```
[student@workstation DO288-apps]$ git checkout main
```

7. Wenn Sie nach Abschluss der nächsten Übung zur ursprünglichen Übung zurückkehren möchten, wechseln Sie zu Ihrem Branch zurück:

```
[student@workstation ~]$ git checkout branch-name
```

Anschließend können Sie mit der ursprünglichen Übung bei dem Schritt fortfahren, bei dem Sie aufgehört haben.



Literaturhinweise

Manpage Git branch

<https://git-scm.com/docs/git-branch>

What is a Git branch?

<https://git-scm.com/book/en/v1/Git-Branching-What-a-Branch-Is>

Git Tools – Stashing

<https://git-scm.com/book/en/v1/Git-Tools-Stashing>