O'REILLY®

GitOps Cookbook

Kubernetes Automation in Practice

Early Release
Raw & Unedited
Compliments of
Red Hat

Natale Vinto &
Alex Soto Bueno

GitOps Cookbook

Kubernetes Automation in Practice

Natale Vinto and Alex Soto Bueno

GitOps Cookbook

by Natale Vinto and Alex Soto Bueno

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

- Acquisitions Editor: John Devins
- Development Editor: Shira Evans
- Production Editor: Kate Galloway
- Interior Designer: David Futato
- Cover Designer: Karen Montgomery
- Illustrator: Kate Dullea
- January 2023: First Edition

Revision History for the Early Release

- 2022-03-08: First Release
- 2022-05-16: Second Release

See http://oreilly.com/catalog/errata.csp?isbn=9781492097471 for release details.

all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

This work is part of a collaboration between O'Reilly and Red Hat. See our statement of editorial independence.

978-1-492-09741-9

Chapter 1. Introduction

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the authors' raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 1st chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at sevans@oreilly.com.

With the advent of practices such as Infrastructure-as-a-Code (IaC), software development is pushed to the boundaries of platforms where to run applications. This becomes more frequent with programmable, API-driven platforms such as public clouds and open source infrastructure solutions. While some years ago developers were only focusing on application source code, today they have also the opportunity to code the infrastructure where their application will run. This gives control and enables automation which significantly reduces lead time.

A good example is with Kubernetes, a popular open-source container workloads orchestration platform and today de-facto standard for running production applications, either on public than private clouds. The openness and extensibility of the platform enables automation, which reduces risks of delivery and increases service quality. Furthermore, this powerful paradigm is extended by another increasingly popular approach called GitOps.

1.1 What is GitOps

GitOps is a methodology and practice that uses git repositories as a single source of truth to deliver infrastructure as code. It takes the pillars and approaches from DevOps culture and provides a framework to start realizing the results. The relationship between DevOps and GitOps is close,

as GitOps has become the popular choice to implement and enhance DevOps and SRE (Site Reliability Engineering).

GitOps is an agnostic approach, and building a GitOps framework can be done with many tools such as git, Kubernetes, and CI/CD solutions. INn synthesis, the three main pillars of GitOps are

- Git is the single source of truth
- Treat everything as code
- Operations through Git workflows

There is an active community around GitOps, and the GitOps Working Group defined a set of GitOps Principles (currently in version 1.0.0) available at OpenGitOps:

1. **Declarative**: A system managed by GitOps must have its desired state expressed declaratively.
2. **Versioned and Immutable**: The desired state is stored in a way that enforces immutability, and versioning and retains a complete version history.
3. **Pulled Automatically**: Software agents automatically pull the desired state declarations from the source.
4. **Continuously Reconciled**: Software agents continuously observe the actual system state and attempt to apply the desired state.

1.2 Why GitOps

Using the common Git-based workflows that developers are familiar with, GitOps expands upon existing processes from application development to deployment, app lifecycle management, and infrastructure configuration.

Every change throughout the application lifecycle is traced in the Git repository and is auditable. This approach is beneficial for both developers and operations teams as it enhances the ability to trace and reproduce issues quickly, improving overall security. One key point is to reduce the risk of unwanted changes (drift) and correct them before they go into production.

Summarizing the benefits of the GitOps adoption in four key aspects:

- ***Standard workflow**: Familiar tools and Git workflows from application development teams
- **Enhanced Security**: Review changes beforehand, detect configuration drifts, and take action
- **Visibility and Audit**: Capturing and tracing any change to clusters through Git history
- **Multi-cluster consistency**: Reliably and consistently configure multiple environments and multiple Kubernetes clusters and deployment

1.3 Kubernetes CI/CD

Continuous Integration (CI) and Continous Delivery (CD) are methods to frequently deliver apps by introducing automation into the stages of app development. CI/CD pipelines are one of the most common use cases for GitOps.

In a typical CI/CD pipeline, submitted code checks the CI process while the CD process checks and applies requirements for things like security, infrastructure as code, or any other boundaries set for the application framework. All code changes are tracked, making updates easy while also providing version control should a rollback be needed. GitOps domain is CD, and it works together with the CI part to deploy apps in multiple environments, as you can see in Figure 1-1.

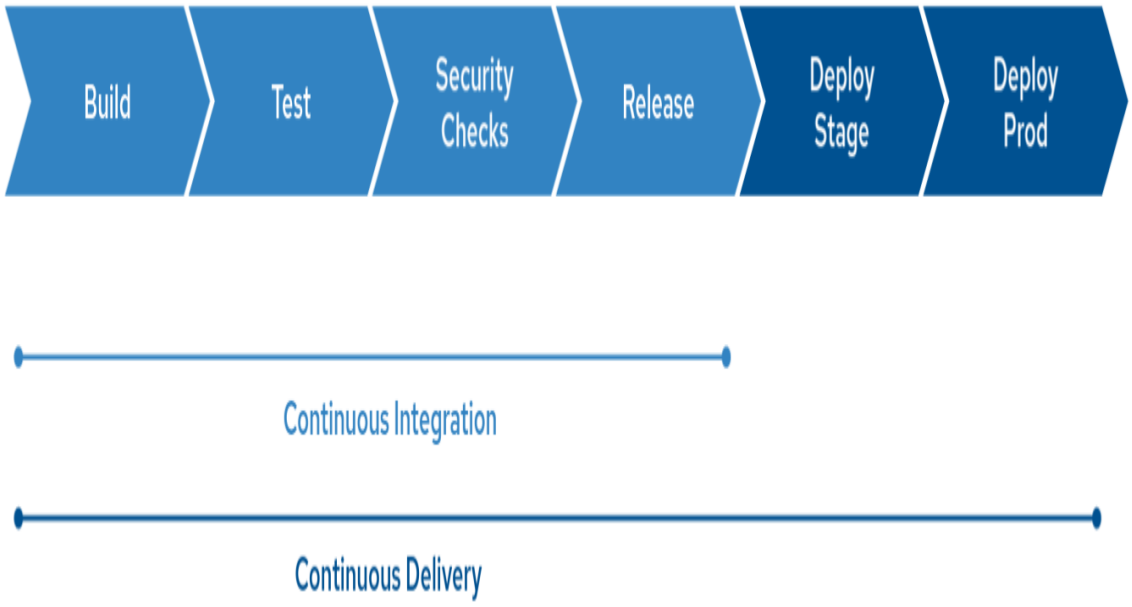


Figure 1-1. Continuous Integration and Continuous Delivery

With Kubernetes, it's easy to implement an in-cluster CI/CD pipeline. You can have CI software create the container image representing your application and store it in a container image registry. Afterward, a git workflow such as a Pull Request can change the Kubernetes manifests illustrating the deployment of your apps and start a CD sync loop as shown in Figure 1-2.

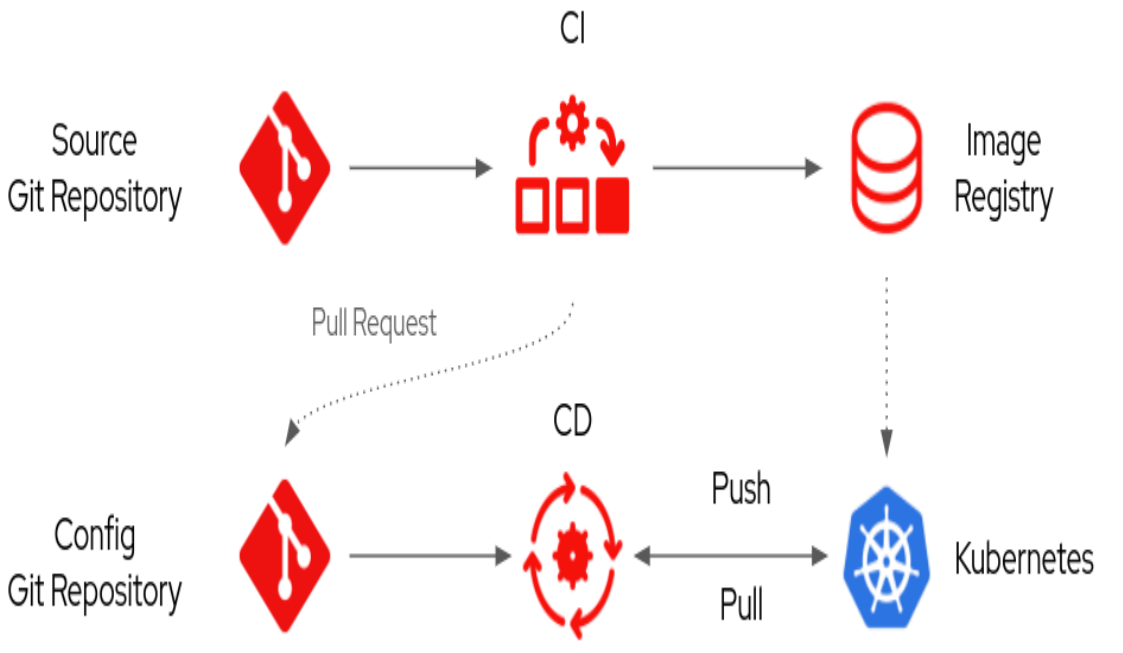


Figure 1-2. Application Deployment Model

This cookbook will show practical recipes for implementing this model on Kubernetes acting as a CI/CD and GitOps platform.

1.4 App deployment with GitOps on Kubernetes

As GitOps is an agnostic, platform-independent approach, the application deployment model on Kubernetes can be either in-cluster or external. An external GitOps tool can use Kubernetes just as a target platform for deploying apps. At the same time, in-cluster approaches run a GitOps engine inside Kubernetes to deploy apps and sync manifests in one or more Kubernetes clusters.

The GitOps engine takes care of the CD part of the CI/CD pipeline and accomplishes a GitOps loop which is composed of four main actions as shown in Figure 1-3:

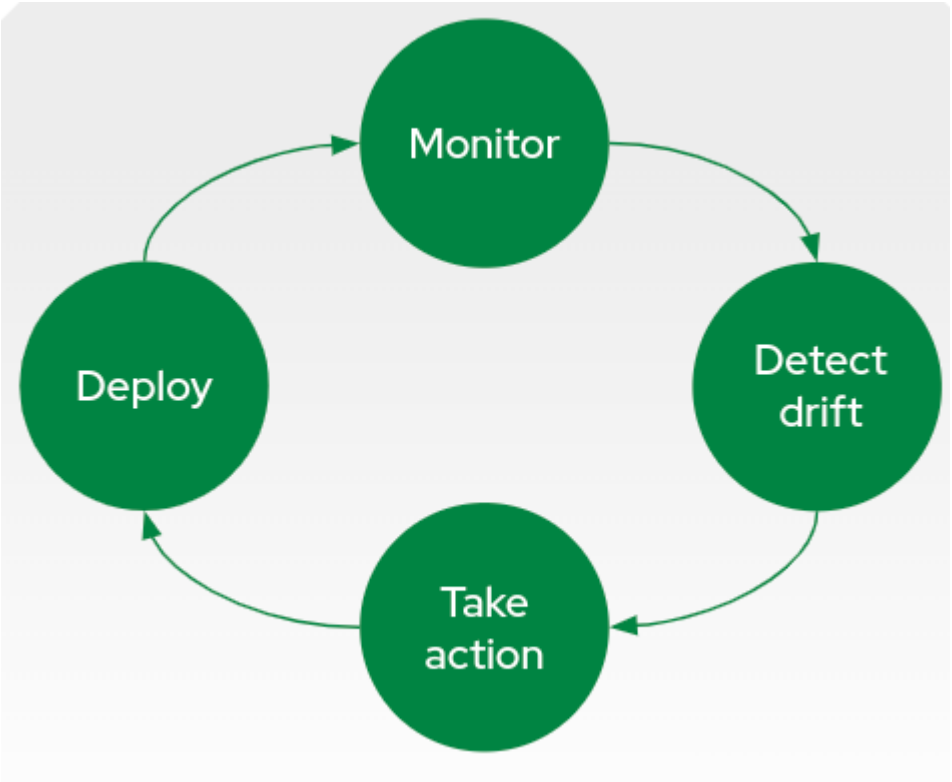


Figure 1-3. GitOps Loop

- **Deploy**: Deploy the manifests from git
- **Monitor**: Monitor either the git repo and the cluster state
- **Detect Drift**: Detect any change from what it's described in git and what it's present in the cluster
- **Take action**: Perform an action that reflects what it's on Git (Rollback, three-way-diff). Git is the source of truth, and any change is performed via a Git Workflow.

In Kubernetes, application deployment using the GitOps approach is comprehensive of at least two git repositories: one for the app source code and one for the Kubernetes manifests describing the app's deployment (Deployment, Service, etc.).

We can summarize the structure of a GitOps project on Kubernetes as in Figure 1-4:

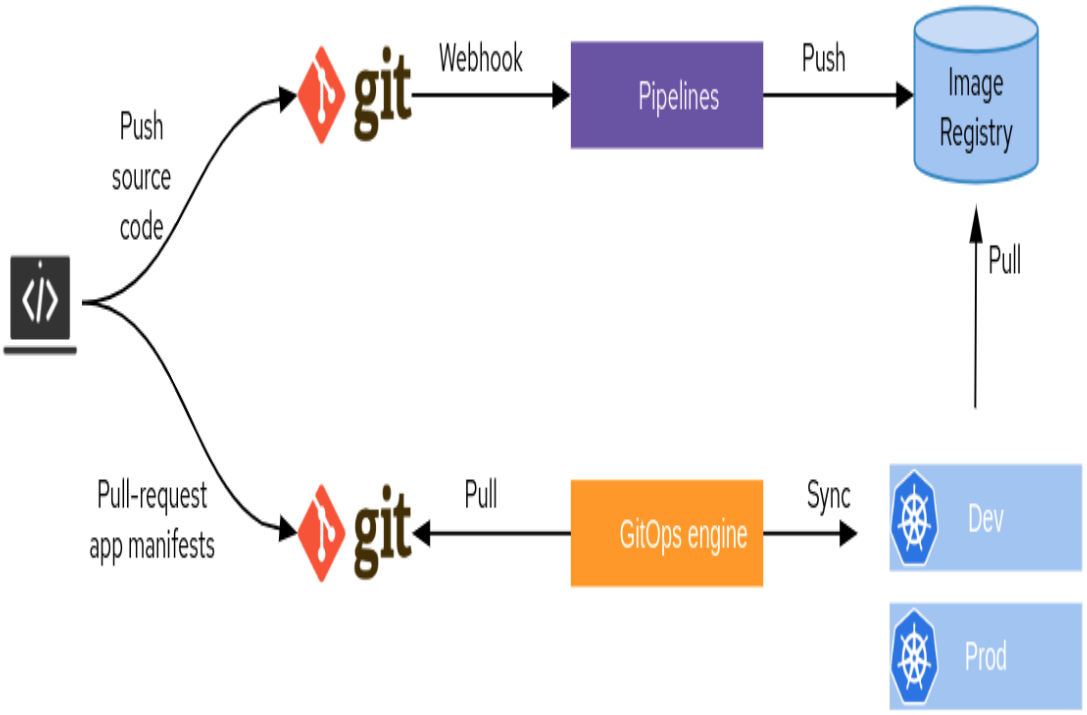


Figure 1-4. Kubernetes GitOps Loop

1. App source code repository
2. CI Pipeline creating a container image
3. Container image registry
4. Kubernetes manifests repository
5. GitOps engine syncing manifests to one or more clusters and detecting drifts

1.5 DevOps and Agility

GitOps is a developer-centric approach to Continuous Delivery and infrastructure operations and a developer workflow through Git for automating processes. As DevOps is complementary to Agile software development, GitOps is complementary to DevOps for infrastructure

automation and application lifecycle management. As you can see in Figure 1-5, it's a developer workflow for automating operations.

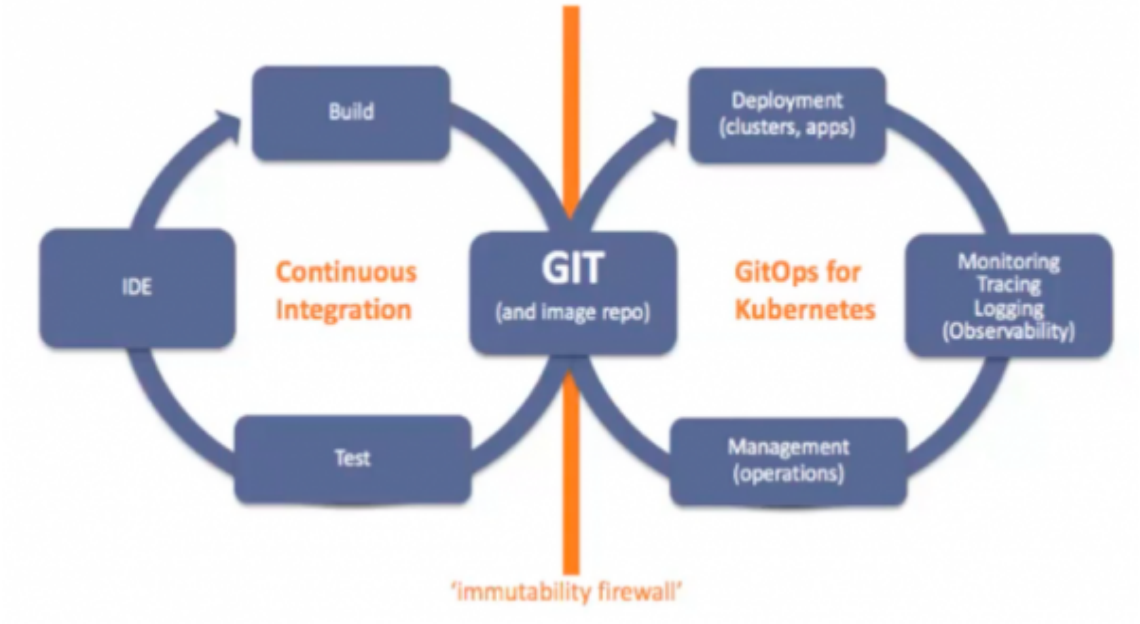


Figure 1-5. GitOps Development Cycle

One of the most critical aspects of the Agile methodology is to reduce the lead time, which is described more abstractly as the time elapsed between identifying a requirement and its fulfillment.[1].

Reducing this time is fundamental and requires a cultural change in IT organizations. Seeing applications live provides developers with a feedback loop to redesign and improve their code and make their projects thrive. As for DevOps, also GitOps requires a cultural adoption in business processes. Every operation, such as application deployment or infrastructure change, is only possible through Git workflows. And sometimes, this means a cultural shift.

The "Teaching Elephants to Dance (and Fly!)" speech from Burr Sutter gives a clear idea of the context. The elephant is where your organization is today. There are phases of change between traditional and modern environments powered by GitOps tools. Some organizations have the

luxury to start from scratch, but the challenge is teaching their lumbering elephant to dance like a graceful ballerina for many businesses.[2]

---

1 *https://www.agilealliance.org/glossary/lead-time/*

2 *http://bit.ly/teachingelephantsxpdays*

Chapter 2. Requirements

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the authors' raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 2nd chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at sevans@oreilly.com.

This book is about GitOps and Kubernetes, and as such, you'll need a container registry to publish the containers built during the book (See Recipe 2.1).

Also, a Git service is required to implement **Git** Ops methodology; you'll learn how to register to public Git services like GitHub or GitLab. (See Recipe 2.2)

Finally, it would be best to have a Kubernetes cluster to run the book examples. Although we'll show you how to install Minikube as a Kubernetes cluster (See Recipe 2.3), and the book is tested with Minikube, any Kubernetes installation might work as well.

Let's prepare your laptop to execute the recipes provided in this book.

2.1 Registering for a container registry

Problem

You want to create an account for a container registry service so you can store generated containers.

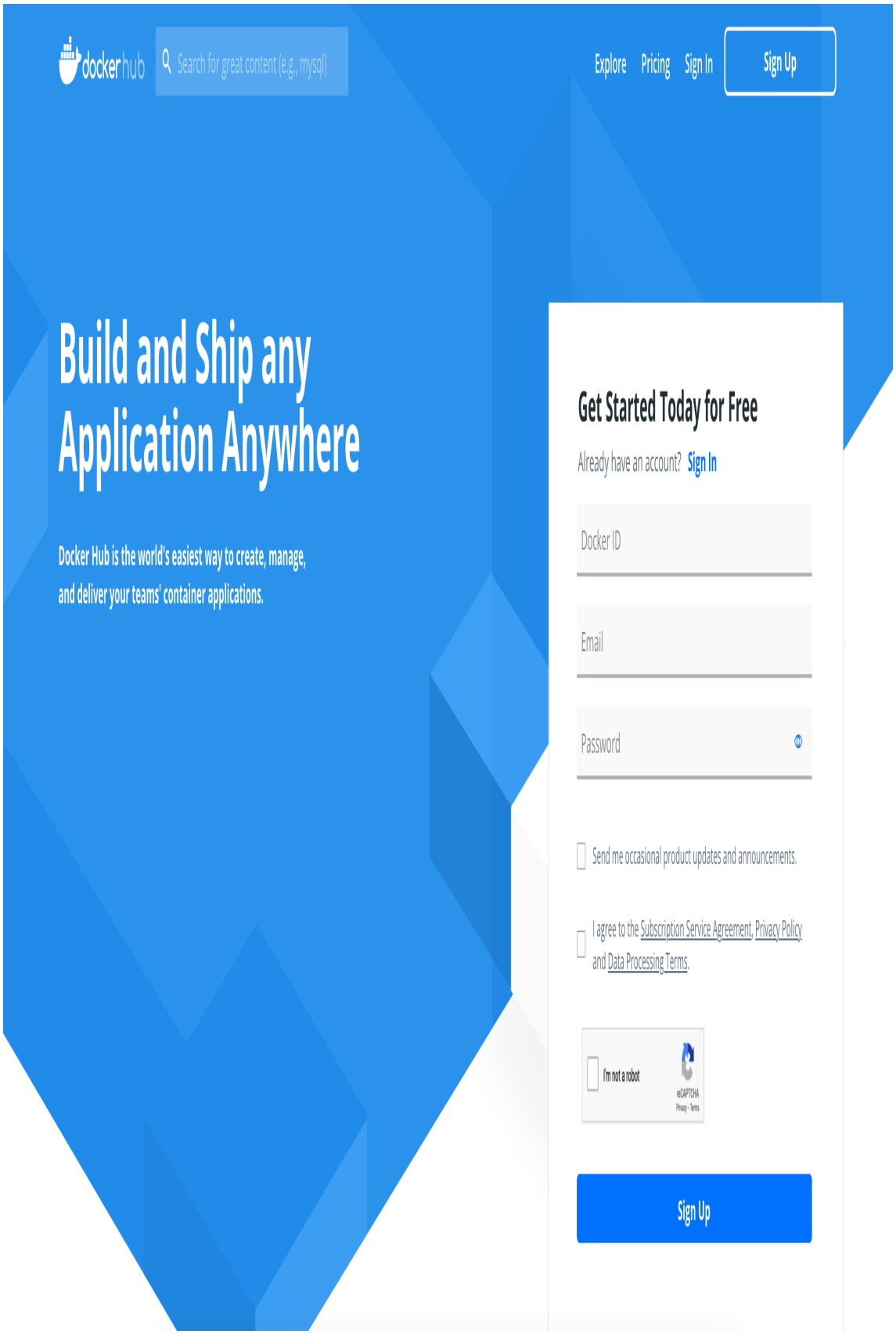
Solution

You may need to publish some containers into a public container registry in this book. Use Docker Hub (`docker.io`) to publish containers.

If you already have an account with `docker.io`, you can skip the following steps, otherwise keep reading to learn how to sign up for an account.

Discussion

Visit the *https://hub.docker.com/* webpage to sign up for an account. The page should be similar as shown in the figure Figure 2-1:


docker hub
Search for great content (e.g., mysql)
Explore
Pricing
Sign In
Sign Up
Build and Ship any Application Anywhere
Docker Hub is the world's easiest way to create, manage, and deliver your teams' container applications.
Get Started Today for Free
Already have an account? Sign In
Docker ID
Email
Password
Send me occasional product updates and announcements.
I agree to the Subscription Service Agreement, Privacy Policy and Data Processing Terms.
I'm not a robot
reCAPTCHA
Privacy - Terms
Sign Up

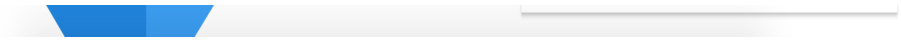


Figure 2-1. DockerHub registration page

When the page is loaded, fill the form setting a *Docker ID*, *Email*, and *Password*, and finally push the **[Sign Up]** button.

When you are registered and your account confirmed, you'll be ready to publish containers under the previous step's _Docker ID _ set.

See Also

Another popular container registry service is `quay.io`. It can be used on the cloud (like `docker.io`) or installed on-premises.

Visit *https://quay.io/* to get more information about Quay. The page should be similar as shown in the figure Figure 2-2:


RED HAT Quay.io
EXPLORE
TUTORIAL
PRICING
search
SIGN IN
Quay.io now supports Red Hat Single Sign-On Services exclusively. If you haven't done so already, you need to link your Quay.io login to a redhat.com account in order to be able to login to the web interface by going to the recovery endpoint. CLI tokens and robot accounts are not impacted. Read more about this change in the FAQ.
Quay [builds, analyzes, distributes] your container images
TRY FOR FREE ON PREMISES
TRY FOR FREE ON CLOUD
podman
docker
rkt

Figure 2-2. Quay registration page

2.2 Registering for a Git registry

Problem

You want to create an account for a Git service so you can store source code in a repository.

Solution

You may need to publish some source code into a public Git service in this book. Use `GitHub` as a Git service to create and fork Git repositories.

If you already have an account with `GitHub`, you can skip the following steps, otherwise keep reading to learn how to sign up for an account.

Discussion

Visit the *https://github.com/* webpage to sign up for an account. The page should be similar as shown in the figure Figure 2-3:

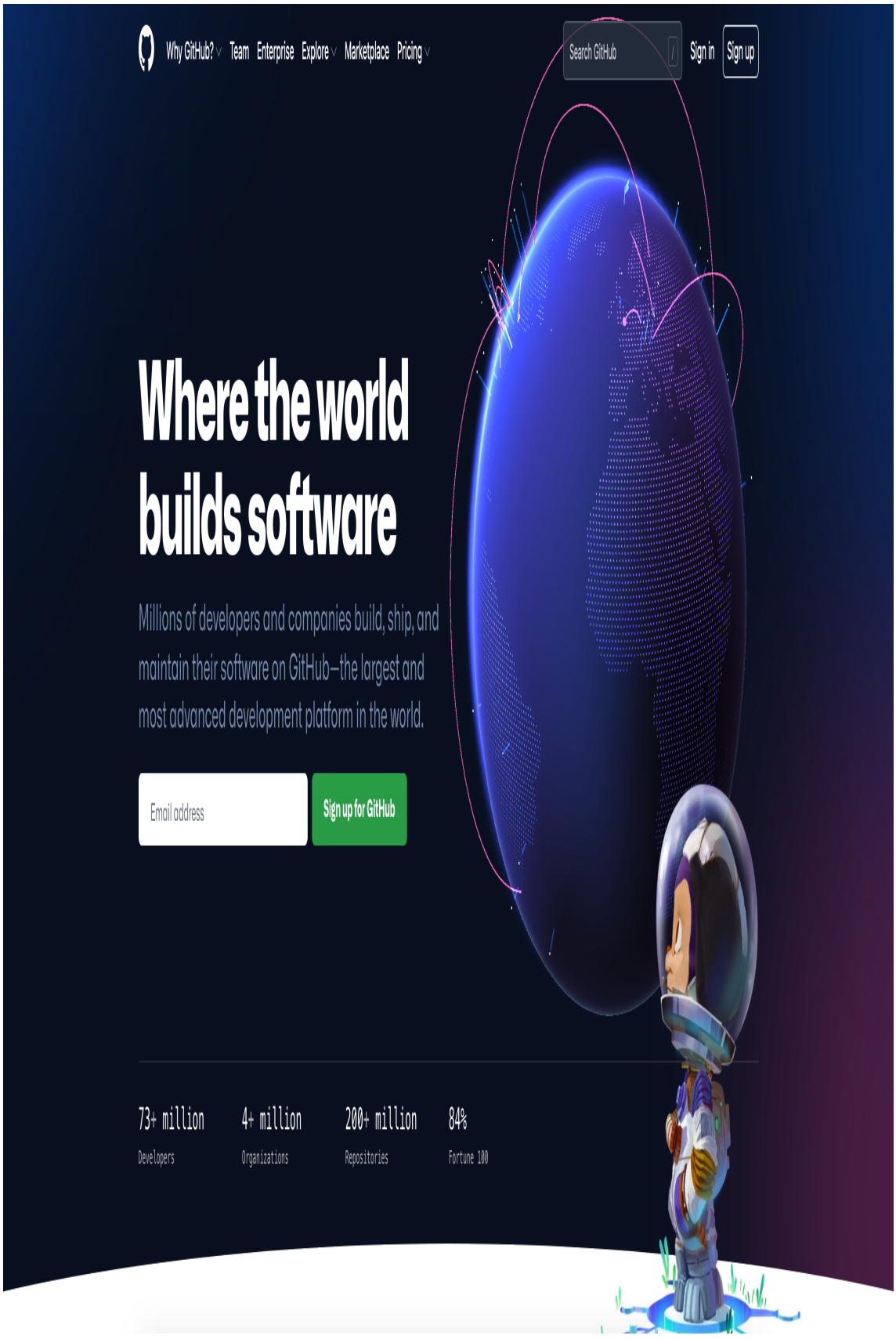

Why GitHub?
Team
Enterprise
Explore
Marketplace
Pricing
Search GitHub
/
Sign in
Sign up
Where the world builds software
Millions of developers and companies build, ship, and maintain their software on GitHub—the largest and most advanced development platform in the world.
Email address
Sign up for GitHub
73+ million
Developers
4+ million
Organizations
200+ million
Repositories
84%
Fortune 100

Figure 2-3. GitHub welcome page to register

When the page is loaded, push the **[Sign In]** button and follow the instructions. The *Sign In* page should be similar as shown in the figure Figure 2-4

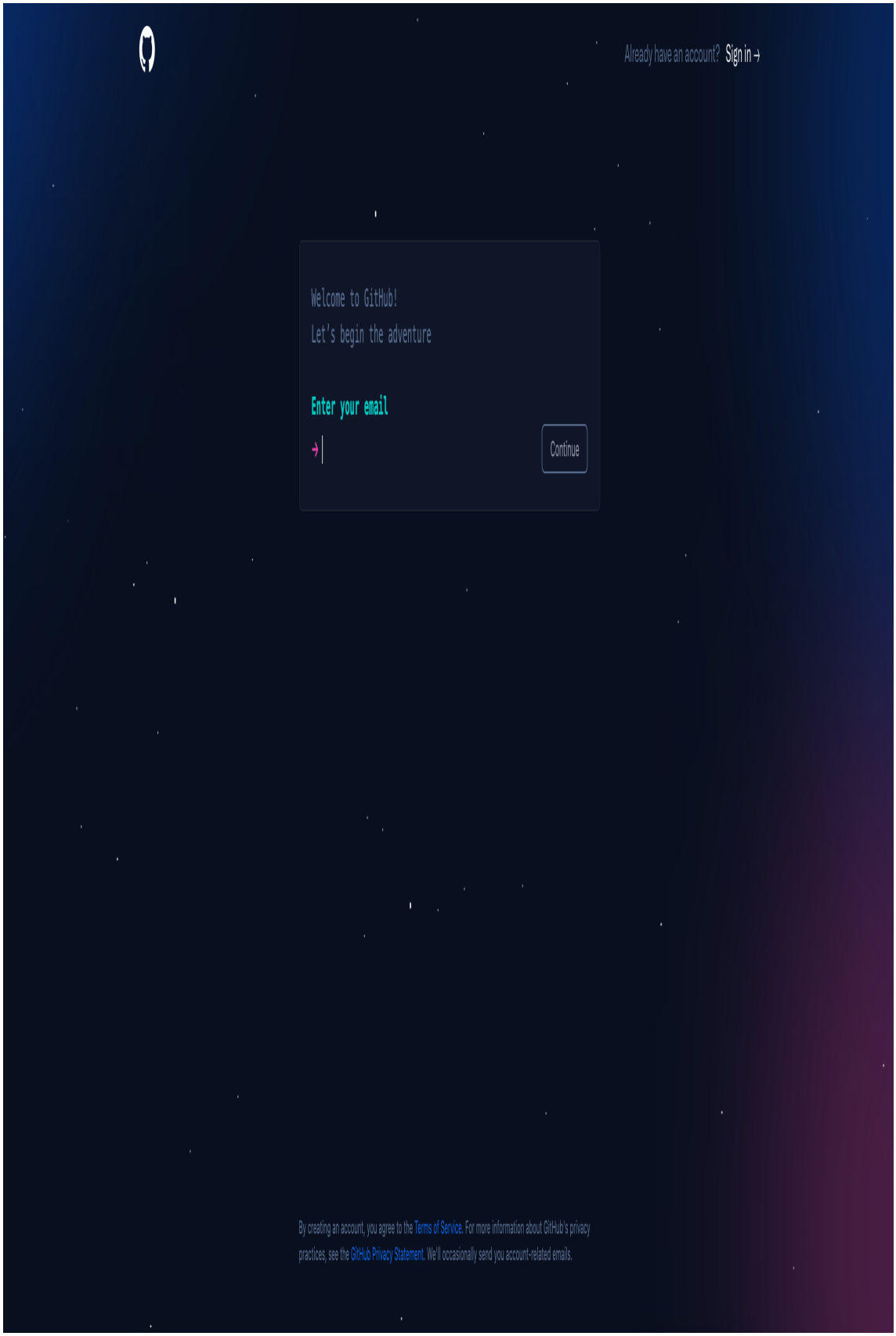

Already have an account? Sign in →
Welcome to GitHub!
Let's begin the adventure
Enter your email
→
Continue
By creating an account, you agree to the Terms of Service. For more information about GitHub's privacy practices, see the GitHub Privacy Statement. We'll occasionally send you account-related emails.

Figure 2-4. Sign In GitHub page

When you are registered and your account confirmed, you'll be ready to start creating or forking Git repositories into your GitHub account.

See Also

Another popular Git service is `GitLab`. It can be used on the cloud or installed on-premises.

Visit *https://about.gitlab.com/* to get more information about GitLab. The page should be similar as shown in the figure Figure 2-5:

GitLab Product v Solutions v Resources v Partners v Pricing Support v

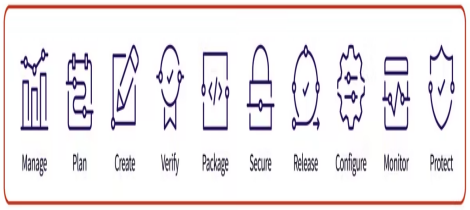
Talk to an expert | Get free trial | Login

The DevOps Platform has arrived.

Deliver software faster with better security and collaboration in a single platform.

Get free trial | Watch demo

The DevOps Platform



Goldman Sachs | SIEMENS | NVIDIA | esa | CLOUD NATIVE COMPUTING FOUNDATION | ticketmaster

Figure 2-5. GitLab welcome page

2.3 Creating a local Kubernetes Cluster

Problem

You want to spin up a Kubernetes cluster locally.

Solution

In this book, you may need a Kubernetes cluster to run most recipes. Use `minikube` to spin up a Kubernetes cluster in your local machine.

Discussion

`minikube` uses container/virtualization technology like Docker, Podman, Hyperkit, Hyper-V, KVM, or VirtualBox to boot up a Linux machine with a Kubernetes cluster installed inside.

For simplicity and to use an installation that might work in most of the platforms, we are going to use VirtualBox as a virtualization system.

To install VirtualBox (if you haven't done it yet), visit *https://www.virtualbox.org/* and click on the **Downloads** link as shown in the figure Figure 2-6.

VirtualBox

search...

Login Preferences

Welcome to VirtualBox.org!

About

Screenshots

Downloads

Documentation

End-user docs

Technical docs

Contribute

Community

VirtualBox is a powerful x86 and AMD64/Intel64 virtualization product for enterprise as well as home use. Not only is VirtualBox an extremely feature rich, high performance product for enterprise customers, it is also the only professional solution that is freely available as Open Source Software under the terms of the GNU General Public License (GPL) version 2. See "About VirtualBox" for an introduction.

Presently, VirtualBox runs on Windows, Linux, Macintosh, and Solaris hosts and supports a large number of guest operating systems including but not limited to Windows (NT 4.0, 2000, XP, Server 2003, Vista, Windows 7, Windows 8, Windows 10), DOS/Windows 3.x, Linux (2.4, 2.6, 3.x and 4.x), Solaris and OpenSolaris, OS/2, and OpenBSD.

VirtualBox is being actively developed with frequent releases and has an ever growing list of features, supported guest operating systems and platforms it runs on. VirtualBox is a community effort backed by a dedicated company: everyone is encouraged to contribute while Oracle ensures the product always meets professional quality criteria.

Hot picks:

- Pre-built virtual machines for developers at Oracle Tech Network
- **Hyperbox** Open-source Virtual Infrastructure Manager project site
- **phpVirtualBox** AJAX web interface project site

News Flash

- Important **May 17th, 2021**
  We're hiring!
  Looking for a new challenge? We're hiring a VirtualBox senior developer in 3D area (Europe/Russia/India).
- New **November 22nd, 2021**
  VirtualBox 6.1.30 released!
  Oracle today released a 6.1 maintenance release which improves stability and fixes regressions. See the Changelog for details.
- New **October 19th, 2021**
  VirtualBox 6.1.28 released!
  Oracle today released a 6.1 maintenance release which improves stability and fixes regressions. See the Changelog for details.
- New **July 28th, 2021**
  VirtualBox 6.1.26 released!
  Oracle today released a 6.1 maintenance release which improves stability and fixes regressions. See the Changelog for details.
- New **July 20th, 2021**
  VirtualBox 6.1.24 released!
  Oracle today released a 6.1 maintenance release which improves stability and fixes regressions. See the Changelog for details.
- New **April 29th, 2021**
  VirtualBox 6.1.22 released!
  Oracle today released a 6.1

Figure 2-6. VirtualBox Home page

Select the package based on the operating system, download it, and finally install it on your computer.

After installing Virtual Box (we used the 6.1.X version), the following step is to download and spin up a cluster using `minikube`.

Visit *https://github.com/kubernetes/minikube/releases/tag/v1.24.0*, unfold the `Assets` section, and download the `minikube` file that matches your platform specification. For example, in the case of an AMD Mac, you should select `minikube-darwin-amd64` as shown in the figure Figure 2-7:

Assets 49

docker-machine-driver-hyperkit	8.35 MB
docker-machine-driver-hyperkit.sha256	65 Bytes
docker-machine-driver-kvm2	11.4 MB
docker-machine-driver-kvm2-1.24.0-0.x86_64.rpm	3.35 MB
docker-machine-driver-kvm2-amd64	11.4 MB
docker-machine-driver-kvm2-amd64.sha256	65 Bytes
docker-machine-driver-kvm2-arm64	11 MB
docker-machine-driver-kvm2-arm64.sha256	65 Bytes
docker-machine-driver-kvm2-x86_64	11.4 MB
docker-machine-driver-kvm2.sha256	65 Bytes
docker-machine-driver-kvm2_1.24.0-0_amd64.deb	5.01 MB
docker-machine-driver-kvm2_1.24.0-0_arm64.deb	4.47 MB
minikube-1.24.0-0.aarch64.rpm	25.1 MB
minikube-1.24.0-0.armv7hl.rpm	25 MB
minikube-1.24.0-0.ppc64le.rpm	24.5 MB
minikube-1.24.0-0.s390x.rpm	26.6 MB
minikube-1.24.0-0.x86_64.rpm	15 MB
minikube-darwin-amd64	65.7 MB
minikube-darwin-amd64.sha256	65 Bytes
minikube-darwin-amd64.tar.gz	30.1 MB

Figure 2-7. Minikube release page

Uncompress the file (if necessary) and copy it with the name `minikube` in a directory accessible by the `PATH` environment variable such as (`/usr/local/bin`) in Linux or MacOS.

With VirtualBox and minikube installed, we can spin up a Kubernetes cluster in the local machine. Let's install Kubernetes version 1.23.0 as it was the latest version when writing the book, although any other previous versions might be used as well.

Run the following command in a terminal window to spin up the Kubernetes cluster:

```
minikube start --kubernetes-version='v1.23.0' \\
--driver='virtualbox' --memory=8196 -p gitops ❶ ❷ ❸
```

❶ Creates a Kubernetes cluster with version 1.23.0

❷ Uses VirtualBox as virtualization tool

❸ Creates a profile name to the cluster to refer to it

And the output lines should be similar to:

```
😄  [gitops] minikube v1.24.0 on Darwin 12.0.1
✨  Using the virtualbox driver based on user configuration
👍  Starting control plane node gitops in cluster gitops ❶
🔥  Creating virtualbox VM (CPUs=2, Memory=8196MB, Disk=20000MB)
...
    > kubeadm.sha256: 64 B / 64 B [--------------------------]
100.00% ? p/s 0s
    > kubelet.sha256: 64 B / 64 B [--------------------------]
100.00% ? p/s 0s
    > kubectl.sha256: 64 B / 64 B [--------------------------]
100.00% ? p/s 0s
    > kubeadm: 43.11 MiB / 43.11 MiB [---------------] 100.00%
3.46 MiB p/s 13s
    > kubectl: 44.42 MiB / 44.42 MiB [---------------] 100.00%
3.60 MiB p/s 13s
    > kubelet: 118.73 MiB / 118.73 MiB [-------------] 100.00%
```

```
6.32 MiB p/s 19s

    ▪ Generating certificates and keys ...
    ▪ Booting up control plane ... ❷
    ▪ Configuring RBAC rules ...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
...

  Verifying Kubernetes components...
🌟  Enabled addons: storage-provisioner, default-storageclass

❗  /usr/local/bin/kubectl is version 1.21.0, which
may have incompatibilites with Kubernetes 1.23.0. ❸
    ▪ Want kubectl v1.23.0? Try 'minikube kubectl -- get pods -A'
🏄  Done! kubectl is now configured to use "gitops" cluster and
 "default" namespace by default ❹
```

❶ Starts the`gitops` cluster

❷ Boot up the Kubernetes cluster control plane

❸ Detects that we have an old `kubectl` tool

❹ Cluster is up and running

To align the Kubernetes cluster and Kubernetes CLI tool version to be the same, we can download the `kubectl` 1.23.0 version running from *https://dl.k8s.io/release/v1.23.0/bin/darwin/amd64/kubectl*.

> **IMPORTANT**
>
> You need to change the `darwin/amd64` to your specific architecture. For example, in Windows might be `windows/amd64/kubectl.exe`.

Copy the `kubectl` CLI tool in a directory accessible by the `PATH` environment variable such as (`/usr/local/bin`) in Linux or MacOS.

See Also

There are other ways to run Kubernetes in a local machine. One that is very popular too is `kind` (*https://kind.sigs.k8s.io/*).

Although the examples of this book should work in any Kubernetes implementation as only standard resources are used, we've only tested with `minikube`.

Chapter 3. Containers

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the authors' raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 3rd chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at sevans@oreilly.com.

Containers are a popular and standard format to package applications, following the Open Container Initiative, an open governance structure for the express purpose of creating open industry standards around container formats and runtimes. The openness of this format ensures portability and interoperability across different operating systems, vendors, platforms, or clouds. Kubernetes runs containerized apps, so before going into the GitOps approach to managing apps on Kubernetes, we provide a list of recipes useful to understand how to package your application as a container image.

The first step around creating images is to use a container engine for packaging your application by building a layered structure containing a base OS and additional layers on top such as runtimes, libraries, applications. Docker is a widespread open-source implementation of a container engine and runtime, and it can generate a container image by specifying a manifest called Dockerfile. (See Recipe 3.1).

Since the format is open, it's possible to create container images with other tools. Docker, a popular container engine, requires the installation and the execution of a *daemon* that can handle all the operations with the container engine. Developers can use an SDK to interact with the Docker daemon or use *dockerless* solutions such as JiB to create container images. (See Recipe 3.2)

If you don’t want to rely on a specific programming language or SDK to build container images, you can use another *daemonless* solution like Buildah (See Recipe 3.3) and Buildpacks (See Recipe 3.4). Those are other growing popular open-source tools for building OCI container images. . By avoiding dependencies from OS, such tools make automation more manageable and portable with Pipelines, as we will see in Chapter 6, or just on Kubernetes.

Kubernetes doesn’t provide a native mechanism for building container images. However, its highly extensible architecture allows interoperability with external tools and the platform’s extensibility to create container images. Shipwright is an open-source framework for building container images on Kubernetes, providing an abstraction that can use tools such as Kaniko, Buildpacks, or Buildah (See Recipes 3.5) to create container images.

At the end of this chapter, you’ll learn how to create OCI compliant container images from a Dockerfile, either from a Docker host than without Docker with tools such as Buildah and Buildpacks that can run outside or inside Kubernetes.

3.1 Building a container using Docker

Problem

You want to create a container image for your application with Docker.

Solution

The first thing you need to do is install Docker.

> **NOTE**
>
> Docker is available for Mac, Windows, and Linux. Download the installer for your operating system and refer to the documentation to start the Docker service.

Developers can create a container image by defining a *Dockerfile*. The best definition for a Dockerfile comes from Docker documentation itself:

"A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image"[1]

Container images present a layered structure, as you can see in Figure 3-1. Each container image provides the foundation layer for a container, and any update is just an additional layer that can be committed on the foundation.

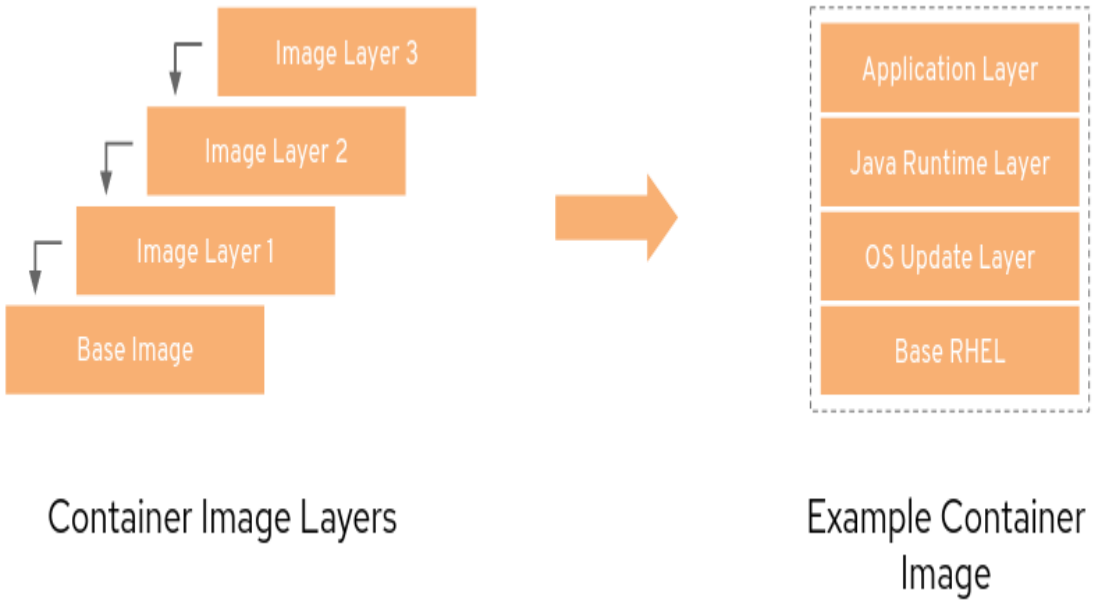


Figure 3-1. Container image layers

You can create a Dockerfile like the one listed below, which will generate a container image for Python apps.

```
FROM registry.access.redhat.com/ubi8/python-39 ❶
ENV PORT 8080 ❷
EXPOSE 8080 ❸
WORKDIR /usr/src/app ❹

COPY requirements.txt ./ ❺
RUN pip install --no-cache-dir -r requirements.txt ❻

COPY . .
```

```
ENTRYPOINT ["python"] ❼
CMD ["app.py"] ❽
```

❶ FROM: always start from a base image as a foundational layer. In this case we start from Universal Base Image (UBI), publicly available based on RHEL 8 with Python 3.9 runtime.

❷ ENV: set an environment variable for the app.

❸ EXPOSE: expose a port to the container network, in this case port TCP 8080.

❹ WORKDIR: set a directory inside the container to work with.

❺ COPY: copy the assets from the source code files on your workstation to the container image layer, in this case, to the WORKDIR.

❻ RUN: run a command inside the container, using the tools available in the layers. In this case, it runs `pip` tool to install dependencies.

❼ ENTRYPOINT: define the entry point for your app inside the container. It can be a binary or a script. In this case, it runs the Python interpreter.

❽ CMD: run a command or pass a parameter. In this case it uses the name of the Python app `app.py`.

You can now create your container image with the following command:

```
docker build -f Dockerfile -t quay.io/gitops-
cookbook/pythonapp:latest
```

Your container image is building now. Docker will fetch existing layers from a public container registry (DockerHub, Quay, Red Hat Registry etc) and add a new layer with the content specified in the Dockerfile. Such layers could also be available locally, if already downloaded, into special storage called *container cache* or *Docker cache*.

```
STEP 1: FROM registry.access.redhat.com/ubi8/python-39
Getting image source signatures
```

```
Copying blob adffa6963146 done
Copying blob 4125bdfaec5e done
Copying blob 362566a15abb done
Copying blob 0661f10c38cc done
Copying blob 26f1167feaf7 done
Copying config a531ae7675 done
Writing manifest to image destination
Storing signatures
STEP 2: ENV PORT 8080
--> 6dbf4ac027e
STEP 3: EXPOSE 8080
--> f78357fe402
STEP 4: WORKDIR /usr/src/app
--> 547bf8ca5c5
STEP 5: COPY requirements.txt ./
--> 456cab38c97
STEP 6: RUN pip install --no-cache-dir -r requirements.txt
Collecting Flask
  Downloading Flask-2.0.2-py3-none-any.whl (95 kB)
     |████████████████████████████████| 95 kB 10.6 MB/s
Collecting itsdangerous>=2.0
  Downloading itsdangerous-2.0.1-py3-none-any.whl (18 kB)
Collecting Werkzeug>=2.0
  Downloading Werkzeug-2.0.2-py3-none-any.whl (288 kB)
     |████████████████████████████████| 288 kB 1.7 MB/s
Collecting click>=7.1.2
  Downloading click-8.0.3-py3-none-any.whl (97 kB)
     |████████████████████████████████| 97 kB 31.9 MB/s
Collecting Jinja2>=3.0
  Downloading Jinja2-3.0.3-py3-none-any.whl (133 kB)
     |████████████████████████████████| 133 kB 38.8 MB/s
STEP 7: COPY . .
--> 3e6b73464eb
STEP 8: ENTRYPOINT ["python"]
--> acabca89260
STEP 9: CMD ["app.py"]
STEP 10: COMMIT quay.io/gitops-cookbook/pythonapp:latest
--> 52e134d39af
52e134d39af013a25f3e44d25133478dc20b46626782762f4e46b1ff6f0243bb
```

Your container image is now available in your Docker cache and ready to be used. You can verify its presence with this command:

```
docker images
```

You should get the list of available container images from the cache in output. Those could be images you have built or downloaded with the

`docker pull` command.

```
REPOSITORY                                    TAG         IMAGE ID
CREATED↳
        SIZE
quay.io/gitops-cookbook/pythonapp             latest
52e134d39af0  6 minutes ago↳
  907 MB
```

Once your image is created, you can consume it locally or push it to a public container registry to be consumed elsewhere, like from a CI/CD pipeline.

You need to first login to your public registry. In this example, we are using Quay:

```
docker login quay.io
```

You should get an output similar to this:

```
Login Succeeded!
```

Then you can push your container image to the registry:

```
docker push quay.io/gitops-cookbook/pythonapp:latest
```

As confirmed, you should get an output similar to this:

```
Getting image source signatures
Copying blob e6e8a2c58ac5 done
Copying blob 3ba8c926eef9 done
Copying blob 558b534f4e1b done
Copying blob 25f82e0f4ef5 done
Copying blob 7b17276847a2 done
Copying blob 352ba846236b done
Copying blob 2de82c390049 done
Copying blob 26525e00a8d8 done
Copying config 52e134d39a done
Writing manifest to image destination
Copying config 52e134d39a [-----------------------------------
-] 0.0b / 5.4KiB
```

```
Writing manifest to image destination
Storing signatures
```

Discussion

Creating container images can be done with Docker in this way from your workstation or any host where the Docker service/daemon is running.

> **TIP**
>
> Additionally, you can use functionalities offered by some public registry such as Quay.io that can directly create the container image from a Dockerfile and store it to the registry.

The build requires access to all layers thus an internet connection to the registries storing base layers is needed, or at least having them in the container cache. Docker has a layered structure where any change to your app is committed on top of the existing layers, so there's no need to download each time all the layers since it will add only deltas for each new change.

> **NOTE**
>
> Container images typically start from a base OS layer such as Fedora, CentOS, Ubuntu, Alpine, etc. However, they can also start from `scratch`, an empty layer for super-minimal images containing only the app's binary. See scratch documentation for more info.

If you want to run your previously created container image, you can do with this command:

```
docker run -p 8080:8080 -ti quay.io/gitops-
cookbook/pythonapp:latest
```

`docker run` has many options to start your container. The most commons are: * -p : it binds the port of the container with the port of the

host running such container * -t : it attaches a TTY to the container * -i : it goes into an interactive mode * -d : it goes in the background, printing a hash that you can use to interact asynchronously with the running container.

The previous command will start your app in the Docker network and bind it to port 8080 of your workstation:

```
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a
production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a
production deployment.
 * Running on http://10.0.2.100:8080/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 103-809-567
```

From a new terminal, try your running container:

```
curl http://localhost:8080
```

You should get an output like this:

```
Hello, World!
```

See Also

- Best practices for writing Dockerfiles
- Manage Docker images

3.2 Building a container using dockerless Jib

Problem

You are a software developer, and you want to create a container image without installing Docker or any additional software on your workstation.

Solution

As discussed in Recipe 3.1, you need to install the Docker engine to create container images. Docker requires permissions to install a service running as a daemon, thus a privileged process in your operating system. Today, there are also available *dockerless* solutions for developers, a popular one is Jib.

Jib is an open-source framework for Java made by Google to build container images OCI-compliant, without the need of Docker or any container runtime. Jib comes as a library that Java developers can import in their Maven or Gradle projects. This means you can create a container image for your app without writing or maintaining any Dockerfiles, delegating this complexity to Jib.

We see the benefits from this approach as the following:[2]

- **Pure Java**: No Docker or Dockerfile knowledge is required. Simply add Jib as a plugin, and it will generate the container image for you. The resulting image is commonly referred to as distro-less since it doesn't inherit from any base image.
- **Speed**: The application is divided into multiple layers, splitting dependencies from classes. There's no need to rebuild the container image like for Dockerfiles; Jib takes care of deploying the layers that changed.
- **Reproducibility**: Unnecessary updates are not triggered because the same contents generate the same image.

The easiest way to kickstart a container image build with Jib on existing Maven is by adding the plugin via command line:

```
mvn compile com.google.cloud.tools:jib-maven-plugin:3.2.0:build -
Dimage=<MY IMAGE>
```

Alternatively, you can do so by adding Jib as a plugin into your `pom.xml`:

```
<project>
  ...
  <build>
    <plugins>
      ...
      <plugin>
        <groupId>com.google.cloud.tools</groupId>
        <artifactId>jib-maven-plugin</artifactId>
        <version>3.2.0</version>
        <configuration>
          <to>
            <image>myimage</image>
          </to>
        </configuration>
      </plugin>
      ...
    </plugins>
  </build>
  ...
</project>
```

In this way, you can also manage other settings such as authentication or parameters for the build.

Let's now add Jib to an existing Java application, a Hello World in Spring Boot that you find in the book's repository.

Run the command below to create a container image without using Docker, and push it directly to a container registry. In this example, we use Quay.io, and we will store the container image at `quay.io/gitops-cookbook/jib-example:latest`, so you will need to provide your credentials for the registry.

```
mvn compile com.google.cloud.tools:jib-maven-plugin:3.2.0:build↵
-Dimage=quay.io/gitops-cookbook/jib-example:latest↵
-Djib.to.auth.username=<USERNAME>↵
-Djib.to.auth.password=<PASSWORD>
```

The authentication here is handled with command-line options, but Jib can manage existing authentication with Docker CLI or read credentials from your `settings.xml`.

The build takes a few moments, and the result is a distro less container image built locally and pushed directly to a registry, in this case, Quay.io.

```
[INFO] Scanning for projects...
[INFO]
[INFO] --------------------------< com.redhat:hello >------------
--------------
[INFO] Building hello 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]-------------------
--------------
...
[INFO] Containerizing application to quay.io/gitops-cookbook/jib-
example...
[INFO] Using credentials from <to><auth> for quay.io/gitops-
cookbook/jib-example
[INFO] The base image requires auth. Trying again for eclipse-
temurin:11-jre...
[INFO] Using base image with digest:↳

sha256:83d92ee225e443580cc3685ef9574582761cf975abc53850c2bc44ec47
d7d943O]
[INFO]
[INFO] Container entrypoint set to [java, -cp, @/app/jib-
classpath-file,↳
 com.redhat.hello.HelloApplication]FO]
[INFO]
[INFO] Built and pushed image as quay.io/gitops-cookbook/jib-
example
[INFO] Executing tasks:
[INFO] [============================] 100,0% complete
[INFO]
[INFO] ------------------------------------------------------------
--------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
--------------
[INFO] Total time:  41.366 s
[INFO] Finished at: 2022-01-25T19:04:09+01:00
[INFO] ------------------------------------------------------------
--------------
```

> **NOTE**
>
> If you have Docker and run the command `docker images`, you won't see this image in your local cache!

Discussion

Your container image is not present in your local cache, as you don't need any container runtime to build images with Jib. You won't see it with `docker images` command, but you pull it from the public container registry afterward, and it will store it in your cache.

This approach is suitable for development velocity and automation, where the CI system doesn't need to have Docker installed on the nodes where it runs. Jib can create the container image without any Dockerfile. Additionally, it can push it to a container registry.

If you also want to store it locally from the beginning, Jib can connect to Docker hosts and do it for you.

You can pull your container image from the registry to try it:

```
docker run -p 8080:8080 -ti quay.io/gitops-cookbook/jib-example

Trying to pull quay.io/gitops-cookbook/jib-example:latest...
Getting image source signatures
Copying blob ea362f368469 done
Copying blob d5cc550bb6a0 done
Copying blob bcc17963ea24 done
Copying blob 9b46d5d971fa done
Copying blob 51f4f7c353f0 done
Copying blob 43b2cdfa19bb done
Copying blob fd142634d578 done
Copying blob 78c393914c97 done
Copying config 346462b8d3 done
Writing manifest to image destination
Storing signatures

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.6.3)

2022-01-25 18:36:24.762  INFO 1 --- [ main]
com.redhat.hello.HelloApplication↳
        : Starting HelloApplication using Java 11.0.13 on
a719cf76f440 with PID 1↳
         (/app/classes started by root in /)
```

```
2022-01-25 18:36:24.765  INFO 1 --- [ main]
com.redhat.hello.HelloApplication↳
        : No active profile set, falling back to default
profiles: default
2022-01-25 18:36:25.700  INFO 1 --- [ main]
o.s.b.w.embedded.tomcat.TomcatWebServer↳
  : Tomcat initialized with port(s): 8080 (http)
2022-01-25 18:36:25.713  INFO 1 --- [ main]
o.apache.catalina.core.StandardService↳
   : Starting service [Tomcat]
2022-01-25 18:36:25.713  INFO 1 --- [ main]
org.apache.catalina.core.StandardEngine↳
  : Starting Servlet engine: [Apache Tomcat/9.0.56]
2022-01-25 18:36:25.781  INFO 1 --- [ main] o.a.c.c.C.[Tomcat].
[localhost].[/]↳
       : Initializing Spring embedded WebApplicationContext
2022-01-25 18:36:25.781  INFO 1 --- [ main]
w.s.c.ServletWebServerApplicationContext↳
 : Root WebApplicationContext: initialization completed in 947 ms
2022-01-25 18:36:26.087  INFO 1 --- [ main]
o.s.b.w.embedded.tomcat.TomcatWebServer↳
  : Tomcat started on port(s): 8080 (http) with context path ''
2022-01-25 18:36:26.096  INFO 1 --- [ main]
com.redhat.hello.HelloApplication↳
        : Started HelloApplication in 1.778 seconds (JVM running
for 2.177)
```

Get the `hello` endpoint:

```
curl localhost:8080/hello
```

```
{"id":1,"content":"Hello, World!"}
```

See Also

- Jib with Quarkus

3.3 Building a container using Buildah

Problem

Sometimes installing or managing Docker is not possible. Dockerless solutions for creating container images are useful in use case such as local

development or CI/CD systems.

Solution

The OCI specification is a open standard and this favors multiple open source implementations for the container engine and the container image building mechanism. Two growing popular examples today are Podman and Buildah.

> **NOTE**
>
> While Podman support is available also on Mac and Windows, Buildah is currently only available on Linux or Linux subsystems such as WSL2 for Windows. See the documentation to install it on your workstation.

Those are two complementary open-source projects and command line tools that work on OCI containers and images, however, they differ in their specialization. While Podman specializes in commands and functions that help you to maintain and modify container images, such as pulling, tagging and pushing, Buildah specializes in building container images. Decoupling functions in different processes is done by design, as the authors wanted to move from the single privileged process Docker model to a lightweighted, rootless, daemonless and decoupled set of tools to improve agility and security.

> **TIP**
>
> Following the same approach, you find Skopeo, a tool used to move container images, and CRI-O, a container engine complaint to Kubernetes container runtime interface for running applications.

Buildah supports Dockerfile format, but its goal is to provide a lower-level interface to build container images without requiring a Dockerfile. Buildah is a *daemonless* solution that can create images inside a container without

mounting the Docker socket. This functionality improves security and portability since it's easy to add Buildah builds on the fly to a CI/CD pipeline where the Linux or Kubernetes nodes don't require Docker installation.

As we discussed, we can create a container image with or without a Dockerfile. Let's now create a simple HTTPD container image without a Dockerfile.

You can start from any base image such as CentOS:

```
buildah from centos
```

You should get an output similar to this:

```
Resolved short name "centos" to a recorded short-name alias↵
 (origin: /etc/containers/registries.conf.d/shortnames.conf)
Getting image source signatures
Copying blob 926a85fb4806 done
Copying config 2f3766df23 done
Writing manifest to image destination
Storing signatures
centos-working-container
```

> **TIP**
>
> Similarly to Docker and `docker images`, you can run the command `buildah containers` to get the list of available images from the container cache. If you also have installed Podman, this is similar to `podman images`.

In this case, the container image ID is `centos-working-container`, and you can refer to it for creating the other layers.

Now let's install `httpd` package inside a new layer:

```
buildah run centos-working-container yum install httpd -y
```

You should get an output similar to this:

```
CentOS Linux 8 - AppStream                              9.0 MB/s |
8.4 MB     00:00
CentOS Linux 8 - BaseOS                                 436 kB/s |
4.6 MB     00:10
CentOS Linux 8 - Extras                                  23 kB/s |
10 kB     00:00
Dependencies resolved.
================================================================
==============
 Package                    Arch    Version          Repository
Size
================================================================
==============
Installing:
 httpd                      x86_64  2.4.37-
43.module_el8.5.0+1022+b541f3b1
Installing dependencies:
 apr                        x86_64  1.6.3-12.el8
 apr-util                   x86_64  1.6.1-6.el8
 brotli                     x86_64  1.0.6-3.el8
 centos-logos-httpd         noarch  85.8-2.el8
 httpd-filesystem           noarch  2.4.37-
43.module_el8.5.0+1022+b541f3b1
 httpd-tools                x86_64  2.4.37-
43.module_el8.5.0+1022+b541f3b1
 mailcap                    noarch  2.1.48-3.el8
 mod_http2                  x86_64  1.15.7-
3.module_el8.4.0+778+c970deab
Installing weak dependencies:
 apr-util-bdb               x86_64  1.6.1-6.el8
 apr-util-openssl           x86_64  1.6.1-6.el8
Enabling module streams:
...
Complete!
```

Now let's copy a welcome HTML page inside the container running HTTPD. You find the source code in this book's repo.

```
<html>
    <head>
        <title>GitOps CookBook example</title>
    </head>
    <body>
        <h1>Hello, World!</h1>
    </body>
</html>
```

```
buildah copy centos-working-container index.html
/var/www/html/index.html
```

For each new layer added, you should get an output with the new container image hash, similarly to the following:

```
78c6e1dcd6f819581b54094fd38a3fd8f170a2cb768101e533c964e04aacab2e

buildah config --entrypoint "/usr/sbin/httpd -DFOREGROUND"
centos-working-container

buildah commit centos-working-container quay.io/gitops-
cookbook/gitops-website
```

You should get an output similar to this:

```
Getting image source signatures
Copying blob 618ce6bf40a6 skipped: already exists
Copying blob eb8c13ba832f done
Copying config b825e91208 done
Writing manifest to image destination
Storing signatures
b825e91208c33371e209cc327abe4f53ee501d5679c127cd71c4d10cd03e5370
```

Your container image is now in the container cache, ready to run or push to another registry.

As mentioned before, Buildah can also create container images from a Dockerfile. Let's make the same container image from the Dockerfile listed below:

```
FROM centos:latest
RUN yum -y install httpd
COPY index.html /var/www/html/index.html
EXPOSE 80
CMD ["/usr/sbin/httpd", "-DFOREGROUND"]

buildah bud -f Dockerfile -t quay.io/gitops-cookbook/gitops-
website

STEP 1: FROM centos:latest
Resolved short name "centos" to a recorded short-name alias↵
```

```
 (origin: /etc/containers/registries.conf.d/shortnames.conf)
Getting image source signatures
Copying blob 926a85fb4806 done
Copying config 2f3766df23 done
Writing manifest to image destination
Storing signatures
STEP 2: RUN yum -y install httpd
CentOS Linux 8 - AppStream                      9.6 MB/s | 8.4 MB
00:00
CentOS Linux 8 - BaseOS                         7.5 MB/s | 4.6 MB
00:00
CentOS Linux 8 - Extras                          63 kB/s |  10 kB
00:00
Dependencies resolved.
...
Complete!
STEP 3: COPY index.html /var/www/html/index.html
STEP 4: EXPOSE 80
STEP 5: CMD ["/usr/sbin/httpd", "-DFOREGROUND"]
STEP 6: COMMIT quay.io/gitops-cookbook/gitops-website
Getting image source signatures
Copying blob 618ce6bf40a6 skipped: already exists
Copying blob 1be523a47735 done
Copying config 3128caf147 done
Writing manifest to image destination
Storing signatures
--> 3128caf1475
3128caf147547e43b84c13c241585d23a32601f2c2db80b966185b03cb6a8025
```

If you have also installed Podman, you can run it this way:

```
podman run -p 8080:80 -ti quay.io/gitops-cookbook/gitops-website
```

Then you can test it by opening the browser on link:http://localhost:8080

Discussion

With Buildah you have the opportunity to create container images from scratch or starting from a Dockerfile. You don't need to install Docker, and everything is designed around security: rootless mechanism, *daemonless* utilities, and more refined control of creating image layers.

Buildah can also build images from scratch, thus an empty layer similar to `FROM scratch` Dockerfile statement. This aspect is useful for creating

very lightweight images containing only the packages needed to run your application, as you can see in Figure 3-2.

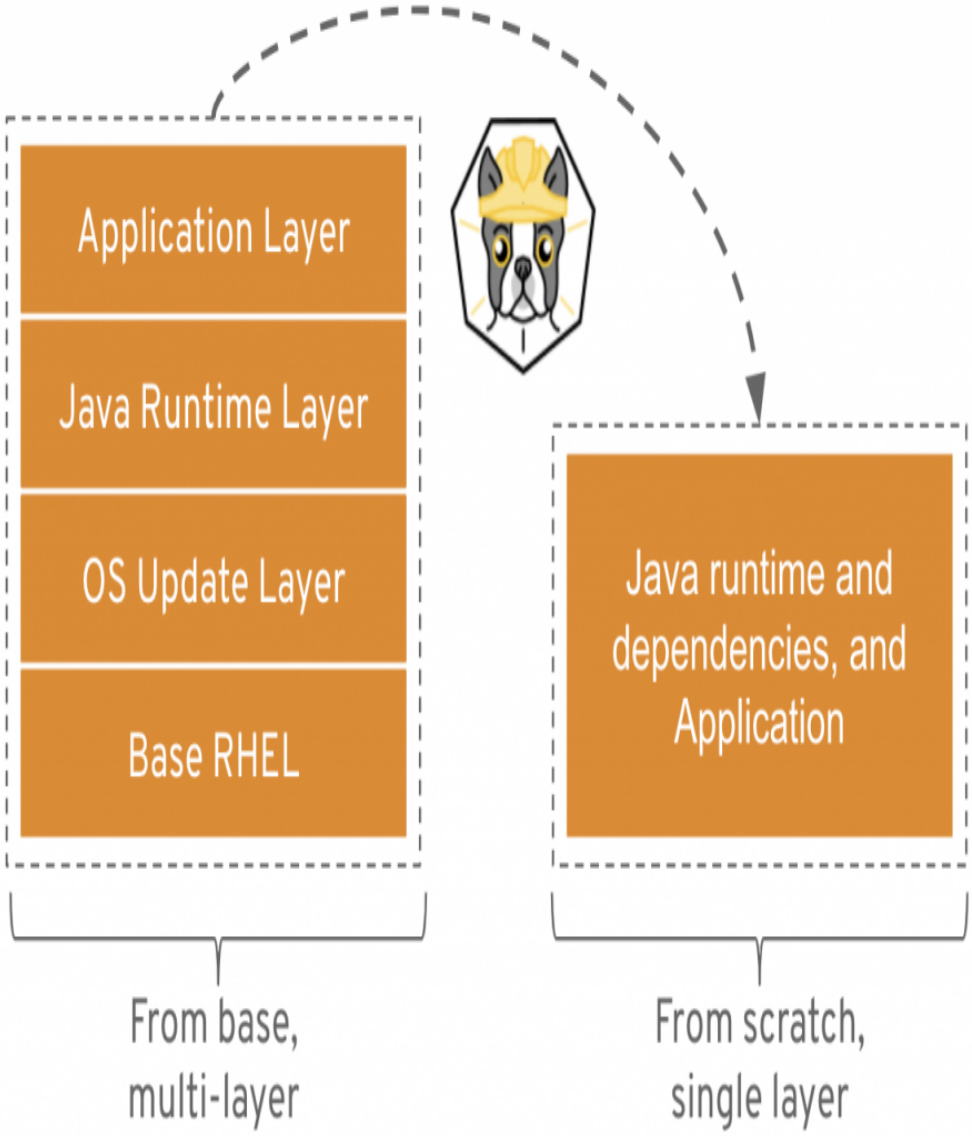


Figure 3-2. Buildah image shrink

A good example use case for a scratch build is considering the development images versus staging or production images. During development, container images may require a compiler and other tools. However, in production, you may only need the runtime or your packages.

See Also

- Running buildah inside a container

3.4 Building a container with Buildpacks

Problem

Creating container image by using Dockerfiles can be challenging at scale. You want a tool complementing Docker that can inspect your application source code to create container images without writing a Dockerfile.

Solution

Cloud Native Buildpacks is an open-source project that provides a set of executables to inspect your app source code and to create a plan to build and run your application.

Buildpack can create OCI-compliant container images without a Dockerfile, starting from the app source code, as you can see in Figure 3-3.

Figure 3-3. Buildpacks builds

This mechanism consists of phases:

- Detection: builpacks tooling will navigate your source code to discover which programming language or framework is used (e.g. POM, NPM files, Python requirements etc.) and assign a suitable buildpack for the build. If anyone is found, the build phase is skipped.
- Building: once a buildpack is found, the source is compiled and Buildpacks creates a container image with the appropriate entry point and startup scripts.

To use Buildpack, you have to download the pack CLI for your operating system (Mac, Windows, Linux), and also have Docker installed.

Now let's start creating our container image with Buildpack from a sample Node.JS app, and you can find the app source code in this book's repository.

The app directory structure contains a `package.json` file, a manifest listing Node.JS package required for this build, which helps Buildpacks understand which buildpack to use.

You can verify it with this command:

```
pack builder suggest
```

You should get an output similar to this:

```
Suggested builders:
        Google:                 gcr.io/buildpacks/builder:v1↳
              Ubuntu 18 base image with buildpacks for .NET, Go,
Java, Node.js,↳
              and Python
        Heroku:                 heroku/buildpacks:18↳
                        Base builder for Heroku-18 stack, based on
ubuntu:18.04 base↳
                        image
        Heroku:                 heroku/buildpacks:20↳
                        Base builder for Heroku-20 stack, based on
ubuntu:20.04 base↳
                        image
        Paketo Buildpacks:      paketobuildpacks/builder:base↳
             Ubuntu bionic base image with buildpacks for Java,
.NET Core, NodeJS,↳
             Go, Python, Ruby, NGINX and Procfile
        Paketo Buildpacks:      paketobuildpacks/builder:full↳
             Ubuntu bionic base image with buildpacks for Java,
.NET Core, NodeJS,↳
             Go, Python, PHP, Ruby, Apache HTTPD, NGINX and
Procfile
        Paketo Buildpacks:      paketobuildpacks/builder:tiny↳
             Tiny base image (bionic build image, distroless-like
run image)↳
             with buildpacks for Java, Java Native Image and Go
```

Now you can decide to pick one of the suggested buildpack. Let's try the `paketobuildpacks/builder:base` which also contains NodeJS runtime.

> **TIP**
>
> Run `pack builder inspect paketobuildpacks/builder:base` to know the exact content of libraries and frameworks available in this buildpack.

```
pack build nodejs-app --builder paketobuildpacks/builder:base
```

The building process should start accordingly, and after a while, it should finish, and you should get an output similar to this:

```
base: Pulling from paketobuildpacks/builder
bf99a8b93828: Pulling fs layer
...
Digest:
sha256:7034e52388c11c5f7ee7ae8f2d7d794ba427cc2802f687dd9650d96a70
ac0772
Status: Downloaded newer image for paketobuildpacks/builder:base
base-cnb: Pulling from paketobuildpacks/run
bf99a8b93828: Already exists
9d58a4841c3f: Pull complete
77a4f59032ac: Pull complete
24e58505e5e0: Pull complete
Digest:
sha256:59aa1da9db6d979e21721e306b9ce99a7c4e3d1663c4c20f74f9b3876c
ce5192
Status: Downloaded newer image for paketobuildpacks/run:base-cnb
===> ANALYZING
Previous image with name "nodejs-app" not found
===> DETECTING
5 of 10 buildpacks participating
paketo-buildpacks/ca-certificates 3.0.1
paketo-buildpacks/node-engine     0.11.2
paketo-buildpacks/npm-install     0.6.2
paketo-buildpacks/node-module-bom 0.2.0
paketo-buildpacks/npm-start       0.6.1
===> RESTORING
===> BUILDING
...
Paketo NPM Start Buildpack 0.6.1
  Assigning launch processes
    web: node server.js

===> EXPORTING
Adding layer 'paketo-buildpacks/ca-certificates:helper'
Adding layer 'paketo-buildpacks/node-engine:node'
Adding layer 'paketo-buildpacks/npm-install:modules'
```

```
Adding layer 'launch.sbom'
Adding 1/1 app layer(s)
Adding layer 'launcher'
Adding layer 'config'
Adding layer 'process-types'
Adding label 'io.buildpacks.lifecycle.metadata'
Adding label 'io.buildpacks.build.metadata'
Adding label 'io.buildpacks.project.metadata'
Setting default process type 'web'
Saving nodejs-app...
*** Images (82b805699d6b):
      nodejs-app
Adding cache layer 'paketo-buildpacks/node-engine:node'
Adding cache layer 'paketo-buildpacks/npm-install:modules'
Adding cache layer 'paketo-buildpacks/node-module-bom:cyclonedx-
node-module'
Successfully built image nodejs-app
```

Now let's run it with Docker:

```
docker run --rm -p 3000:3000 nodejs-app
```

You should get an output similar to this:

```
Server running at http://0.0.0.0:3000/
```

See it in the running:

```
curl http://localhost:3000/
```

You should get an output similar to this:

```
Hello Buildpacks!
```

Discussion

Cloud Native Buildpacks is an incubating project in the Cloud Native Computing Foundation (CNCF), and it supports both Docker and Kubernetes. On Kubernetes, it can be used with Tekton, a Kubernetes-native CI/CD system that can run buildpacks as a Tekton `Task` to create container images. It recently adopted the Boson Project to provide a

Functions-as-a-Service (FaaS) experience on Kubernetes with Knative, by enabling the build of functions via buildpacks.

See Also

- Builpacks Tekton task
- Boson project's buildpacks

3.5 Building a container using Shipwright and Kaniko in Kubernetes

Problem

You need to create a container image, and you want to do it with Kubernetes.

Solution

Kubernetes is well-known as a container orchestration platform to deploy and manage apps. However, it doesn't much for building container images out-of-the-box. Indeed, according to Kubernetes documentation:

"(Kubernetes) Does not deploy source code and does not build your application. Continuous Integration, Delivery, and Deployment (CI/CD) workflows are determined by organization cultures and preferences as well as technical requirements."[3]

As mentioned, one standard option is to rely on CI/CD systems for this purpose, like Tekton (See Chapter 6). Another option is to use a framework to manage builds with many underlying tools, such as the one we discussed in the previous recipes. One example is Shipwright.

Shipwright is an extensible framework for building container images on Kubernetes. It supports popular tools such as Buildah, Cloud Native Buildpacks, or Kaniko. It uses Kubernetes-style APIs, and it runs workloads using Tekton.

The benefit for developers is a simplified approach for building container images, by defining a minimal YAML file that does not require any previous knowledge of containers or container engines. This approach makes this solution agnostic and highly integrated with the Kubernetes API ecosystem.

The first thing to do is to install it to your Kubernetes cluster, say KinD or Minikube (See Chapter 2), following the documentation or from OperatorHub.io.

> **TIP**
>
> Using Operators and Operator Lifecycle Manager(OLM) gives consistency for installing/uninstalling software on Kubernetes, along with dependencies management and lifecycle control. For instance, the Tekton Operator dependency is automatically resolved and installed if you install Shipwright via the Operator. Check OLM documentation go into detail with this approach.

Let's follow the standard procedure from the documentation.

First you need to install Tekton dependency. At the time of writing this book in version `0.30.0`:

```
kubectl apply -f \
  https://storage.googleapis.com/tekton-
releases/pipeline/previous/v0.30.0/release.yaml
```

Then you install Shipwright. At the time of writing this book in version `0.7.0`:

```
kubectl apply -f \
  https://github.com/shipwright-
io/build/releases/download/v0.7.0/release.yaml
```

Finally you install Shipwright build strategies:

```
kubectl apply -f \
  https://github.com/shipwright-
io/build/releases/download/v0.7.0/release.yaml
```

Once you have installed Shipwright, you can start creating your container image build using one of these tools:

- Kaniko
- Cloud Native Buildpacks
- BuildKit
- Buildah

Let's explore Kaniko.

Kaniko is another *dockerless* solution to build container images from a Dockerfile inside a container or Kubernetes cluster. Shipwright brings additional APIs to Kubernetes to use tools such as Kaniko to create container images, acting as an abstract layer that can be considered an extensible building system for Kubernetes.

Let's explore these APIs in terms of Customer Resource Definition (CRD):

- `ClusterBuildStrategy`: represents the type of build to execute.
- `Build`: represents the build. It includes the specification of one ClusterBuildStrategy
- `BuildRun`: represents a running build. The build starts when this object is created.

Run the following command to check all available `ClusterBuildStrategy` (CBS) objects:

```
kubectl get cbs
```

You should get a list of available CBS to consume:

```
NAME                     AGE
buildah                  26s
buildkit                 26s
buildpacks-v3            26s
buildpacks-v3-heroku     26s
kaniko                   26s
kaniko-trivy             26s
ko                       26s
```

```
source-to-image          26s
source-to-image-redhat   26s
```

> **NOTE**
>
> This CRD is cluster-wide, available for all namespaces. If you don't see any items, please install the Shipwright build strategies as discussed before.

Shipwright will generate a container image on the Kubernetes nodes container cache, and then it can push it to a container registry.

You need to provide the credentials to push the image to the registry in the form of a Kubernetes `Secret`. For example, if you use Quay you can create one like the following:

> **TIP**
>
> With Quay, you can use an encrypted password instead of using your account password. See the documentation for more details.

```
REGISTRY_SERVER=quay.io
REGISTRY_USER=<your_registry_user>
REGISTRY_PASSWORD=<your_registry_password>
EMAIL=<your_email>
kubectl create secret docker-registry push-secret \
    --docker-server=$REGISTRY_SERVER \
    --docker-username=$REGISTRY_USER \
    --docker-password=$REGISTRY_PASSWORD  \
    --docker-email=$EMAIL
```

Now let's create the Build object that uses Kaniko to containerize a NodeJS sample app. You find the source code in this book's repository:

```
apiVersion: shipwright.io/v1alpha1
kind: Build
metadata:
  name: buildpack-nodejs-build
spec:
  source:
```

```
    url: https://github.com/shipwright-io/sample-nodejs ❶
    contextDir: docker-build ❷
  strategy:
    name: kaniko ❸
    kind: ClusterBuildStrategy
  output:
    image: quay.io/gitops-cookbook/sample-nodejs:latest ❹
    credentials:
      name: push-secret ❺
```

❶ Repository where to grab the source code.

❷ The directory where the source code is present.

❸ The ClusterBuildStrategy to use.

❹ The destination of the resulting container image

❺ The secret to use to authenticate to the container registry and push the image.

```
kubectl create -f build.yaml
```

You should get an output similar to this:

```
build.shipwright.io/kaniko-nodejs-build created
```

```
kubectl get builds
```

You should get an output similar to the following:

```
NAME                  REGISTERED   REASON      BUILDSTRATEGYKINDl
      BUILDSTRATEGYNAME   CREATIONTIME
kaniko-nodejs-build   True         Succeeded
ClusterBuildStrategyl
   kaniko              13s
```

At this point, your Build is `REGISTERED`, but it's not started yet. Let's create the following object in order to start it:

```
apiVersion: shipwright.io/v1alpha1
kind: BuildRun
```

```
metadata:
  generateName: kaniko-nodejs-buildrun-
spec:
  buildRef:
    name: kaniko-nodejs-build
```

If you check the list of running Pods, you should see one creating:

```
kubectl get pods
```

```
NAME                                                READY   STATUS
RESTARTS↳
      AGE
kaniko-nodejs-buildrun-b9mmb-qbrgl-pod-dk7xt   0/3
PodInitializing   0↳
             19s
```

When the `STATUS` changes, the build will start, and you can track the progress by checking the logs from the containers used by this Pod to run the build in multiple steps:

- step-source-default: the first step, used to get the source code
- step-build-and-push: the step to run the build, either from source code or from a Dockerfile like in this case with Kaniko
- step-results: the result of the build

Let's check the logs of the building phase:

```
kubectl logs -f kaniko-nodejs-buildrun-b9mmb-qbrgl-pod-dk7xt -c
step-build-and-push
```

```
INFO[0001] Retrieving image manifest ghcr.io/shipwright-
io/shipwright-samples/node:12
INFO[0001] Retrieving image ghcr.io/shipwright-io/shipwright-
samples/node:12↳
 from registry ghcr.io
INFO[0002] Built cross stage deps: map[]
INFO[0002] Retrieving image manifest ghcr.io/shipwright-
io/shipwright-samples/node:12
INFO[0002] Returning cached image manifest
INFO[0002] Executing 0 build triggers
INFO[0002] Unpacking rootfs as cmd COPY . /app requires it.
INFO[0042] COPY . /app
```

```
INFO[0042] Taking snapshot of files...
INFO[0042] WORKDIR /app
INFO[0042] cmd: workdir
INFO[0042] Changed working directory to /app
INFO[0042] No files changed in this command, skipping
snapshotting.
INFO[0042] RUN     pwd &&     ls -l &&     npm install &&↳
     npm run print-http-server-version
INFO[0042] Taking snapshot of full filesystem...
INFO[0052] cmd: /bin/sh
INFO[0052] args: [-c pwd &&     ls -l &&     npm install &&↳
     npm run print-http-server-version]
INFO[0052] Running: [/bin/sh -c pwd &&     ls -l &&     npm
install &&↳
     npm run print-http-server-version]
/app
total 44
-rw-r--r-- 1 node node   261 Jan 27 14:29 Dockerfile
-rw-r--r-- 1 node node 30000 Jan 27 14:29 package-lock.json
-rw-r--r-- 1 node node   267 Jan 27 14:29 package.json
drwxr-xr-x 2 node node  4096 Jan 27 14:29 public
npm WARN npm-simple-renamed@0.0.1 No repository field.
npm WARN npm-simple-renamed@0.0.1 No license field.

added 90 packages from 40 contributors and audited 90 packages in
6.405s

10 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities


> npm-simple-renamed@0.0.1 print-http-server-version /app
> serve -v

13.0.2
INFO[0060] Taking snapshot of full filesystem...
INFO[0062] EXPOSE 8080
INFO[0062] cmd: EXPOSE
INFO[0062] Adding exposed port: 8080/tcp
INFO[0062] CMD ["npm", "start"]
INFO[0070] Pushing image to quay.io/gitops-cookbook/sample-
nodejs:latest
INFO[0393] Pushed image to 1 destinations
```

The image is built and pushed to the registry, and you can check the result from this command as well:

```
kubectl get buildruns
```

And on your registry as shown in Figure 3-4.


RED HAT Quay.io
EXPLORE
APPLICATIONS
REPOSITORIES
TUTORIAL
search
bluesma...
Quay.io now supports Red Hat Single Sign-On Services exclusively. If you haven't done so already, you need to link your Quay.io login to a redhat.com account in order to be able to login to the web interface by going to the recovery endpoint. CLI tokens and robot accounts are not impacted. Read more about this change in the FAQ.
Repositories
gitops-cookbook / sample-nodejs
Repository Tags
Compact
Expanded
1 - 1 of 1
Filter Tags...
TAG
LAST MODIFIED
SECURITY SCAN
SIZE
EXPIRES
MANIFEST
latest
10 minutes ago
Queued
337.7 MB
Never
SHA256 8c09c31bafab

Figure 3-4. Image pushed to Quay

Discussion

Shipwright provides a convenient way to create container images on Kubernetes, and its agnostic approach makes it robust and interoperable. The project aims at being the Build API for Kubernetes, providing an easier path for developers to automate on Kubernetes. As Tekton runs under the hood creating, Shipwright also makes transitioning from micro-pipelines to extended pipelines workflows on Kubernetes easier.

As a reference, if you would like to create a build with Buildah instead of Kaniko, it's just a `ClusterBuildStrategy` change in your `Build` object:

```
apiVersion: shipwright.io/v1alpha1
kind: Build
metadata:
  name: buildpack-nodejs-build
spec:
  source:
    url: https://github.com/shipwright-io/sample-nodejs
    contextDir: source-build ❶
  strategy:
    name: buildah ❷
    kind: ClusterBuildStrategy
  output:
    image: quay.io/gitops-cookbook/sample-nodejs:latest
    credentials:
      name: push-secret
```

❶ As we discussed previously in 3.3, Buildah can create the container image from the source code. It doesn't need a Dockerfile

❷ Selecting Buildah as ClusterBuildStrategy

3.6 Final Thoughts

The container format is de facto standard for packaging applications, and today many tools help create container images. Developers can create images with Docker or with other tools and frameworks and then use the same with any CI/CD system to deploy their apps to Kubernetes.

While Kubernetes per se doesn't build container images, some tools interact with the Kubernetes API ecosystem to add this functionality. This aspect improves development velocity and consistency across environments, delegating this complexity to the platform.

In the following chapters, you will see how to control the deployment of your containers running on Kubernetes with tools such as Kustomize or Helm, and then how to add automation to support highly scalable workloads with CI/CD and GitOps.

---

[1] Docker documentation: *https://docs.docker.com/engine/reference/builder/*

[2] Presentation about Jib: *https://speakerdeck.com/coollog/build-containers-faster-with-jib-a-google-image-build-tool-for-java-applications?slide=36*

[3] *https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/#what-kubernetes-is-not*

About the Authors

Natale Vinto is a Software Engineer with more than 10 years of expertise on IT and ICT technologies and a consolidated background on Telecommunications and Linux operating systems. As a Solution Architect with a Java development background, he spent some years as EMEA Specialist Solution Architect for OpenShift at Red Hat. Today Natale is Developer Advocate for OpenShift at Red Hat, helping people within communities and customers having success with their Kubernetes and Cloud Native strategy. You can follow more frequent updates on his Twitter feed and connect to him on LinkedIn.

Alex Soto is a Director of Developer Experience at Red Hat. He is passionate about the Java world, software automation, and he believes in the open-source software model. Alex is the co-author of Testing Java Microservices, Quarkus Cookbook, Securing Kubernetes Secrets and contributor to several open-source projects. A Java Champion since 2017, he is also an international speaker and teacher at Salle URL University. . You can follow more frequent updates on his Twitter feed and connect to him on LinkedIn.