

Creating Custom Container Images





Designing Custom Container Images

After completing this section, you will be:

Describe

- the approaches for creating custom container images.

Find

- existing Dockerfiles to use as a starting point for creating a custom container image.

Define the role played by

- Red Hat Software Collections Library (RHSC) in designing container images

Describe the

- Source-to-Image (S2I) alternative to Dockerfiles.



Reusing Existing Dockerfiles

- Build container image
- Customization
- Easy to share
- Version control
- Reusability
- Extendibility

Case scenarios creating Dockerfile

- Add new runtime libraries, such as database connectors.
- Include organization-wide customization such as SSL certificates and authentication providers.
- Add internal libraries to be shared as a single image layer by multiple container images for different applications.
- Trim the container image by removing unused material (such as man pages, or documentation found in `/usr/share/doc`).
- Lock either the parent image or some included software package to a specific release to lower risk related to future software updates.

Working with the Red Hat Software Collections Library

- Solution for Developers for latest development tools
- Stable environment
- Reliable Images
- Regularly updated
- All RHEL subscribers

Finding Dockerfiles from RHSCl

- RHSCl is the source of most container images
- RHSCl-dockerfiles package available from RHSCl repository
- CentOS-based container images
 - <https://github.com/sclorg?q=-container>
- Most RHSCl container images support Source-to-Image (S2I)

Container Images in Red Hat Container Catalog (RHCC)

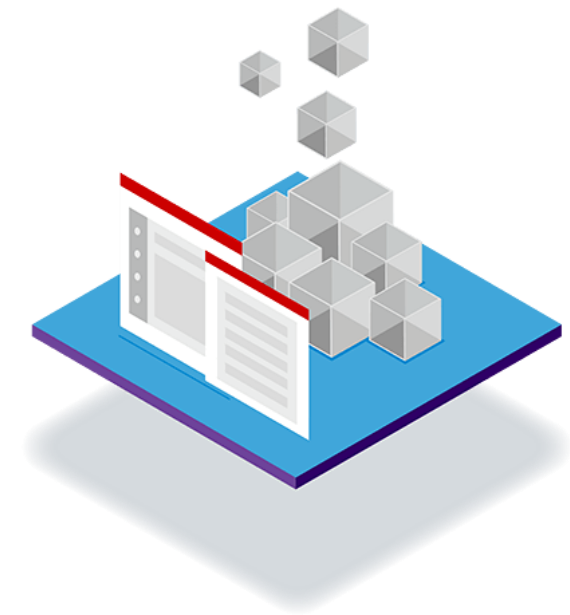
- Trusted containers
- Repository for
 - Reliable
 - Tested
 - Certified
 - Curated collection of container images
- Built with RHEL and related systems
- Quality-assurance process
- Built to avoid known security vulnerabilities
- Updated regularly

Searching for Images Using Quay.io

- Advanced container repository
- Optimized for team collaboration
- Information page
- Vulnerability scan
- Plan
- Free plan
- Public vs Private repository

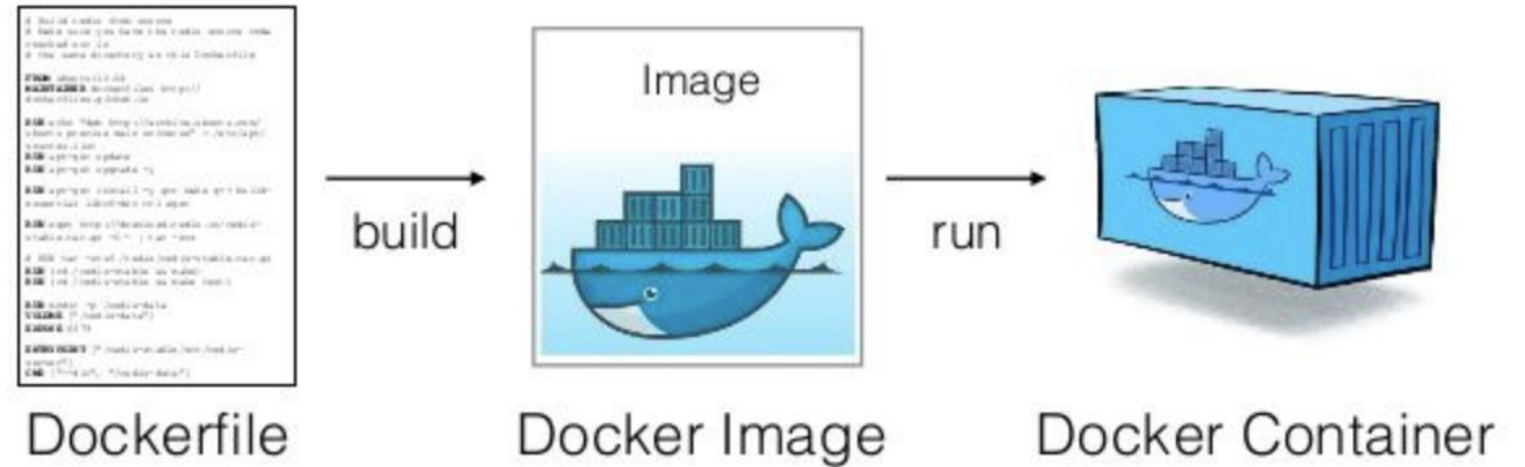


Red Hat
Quay.io



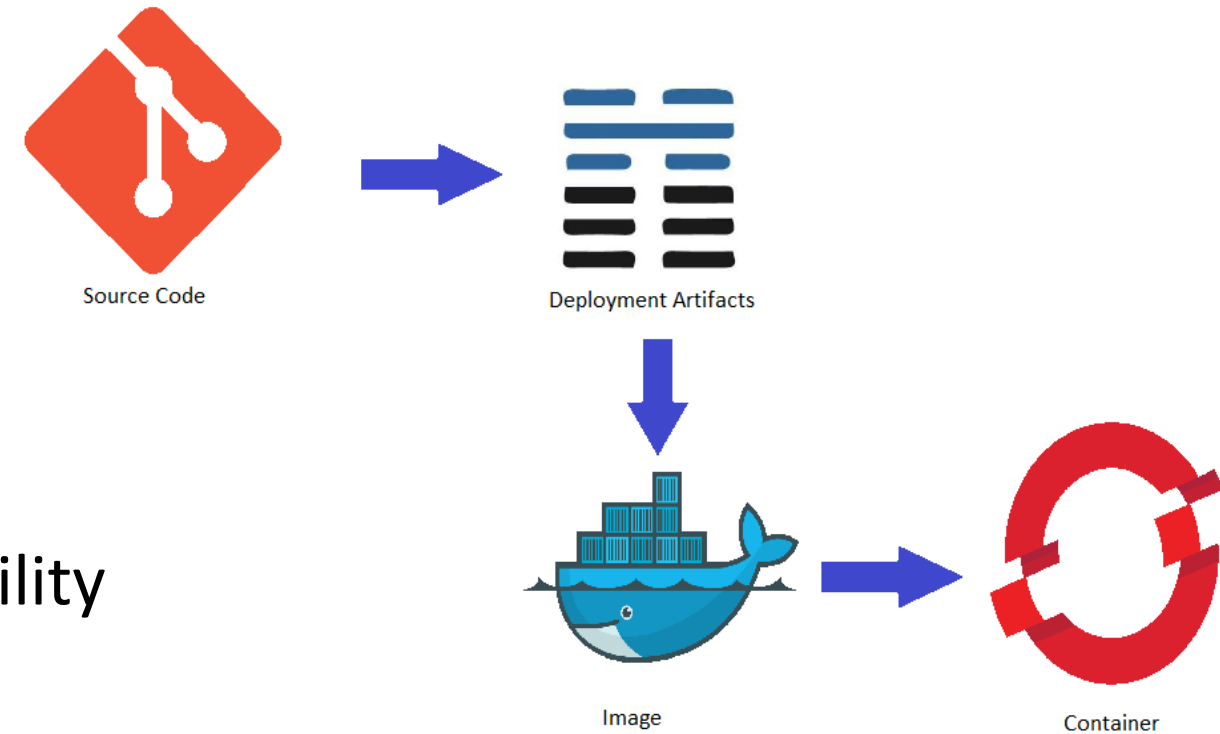
Finding Dockerfiles on Docker Hub

- Community hub for docker container images
- Risks for quality and security
- Evaluate images carefully
- Look for Dockerfiles via github link



Describing How to use the OpenShift Source-to-Image Tool

- Alternatives to building Dockerfile from scratch
- Feature from OpenShift or as standalone s2i utility
- Builder image
 - Programming language
 - Development tools (compilers, package managers)
- Build custom workflows:
 1. Start container from builder image
 2. Fetch application source code
 3. Build application binary
 4. Clean up and save container into registry



Quiz 1

Which method for creating container images is recommended by the containers community?

- a) Run commands inside a basic OS container, commit the container, and save or export it as a new container image.
- b) Run commands from a Dockerfile and push the generated container image to an image registry.
- c) Create the container image layers manually from tar files.
- d) Run the podman build command to process a container image description in YAML format.

Quiz 1

Which method for creating container images is recommended by the containers community?

- a) Run commands inside a basic OS container, commit the container, and save or export it as a new container image.
- b) Run commands from a Dockerfile and push the generated container image to an image registry.
- c) Create the container image layers manually from tar files.
- d) Run the podman build command to process a container image description in YAML format.

Quiz 2

What are two advantages of using the standalone S2I process as an alternative to Dockerfiles? (Choose two.)

- a) Requires no additional tools apart from a basic Podman setup.
- b) Creates smaller container images, having fewer layers.
- c) Reuses high-quality builder images.
- d) Automatically updates the child image as the parent image changes (for example, with security fixes).

Quiz 2

What are two advantages of using the standalone S2I process as an alternative to Dockerfiles? (Choose two.)

- a) Requires no additional tools apart from a basic Podman setup.
- b) Creates smaller container images, having fewer layers.
- c) Reuses high-quality builder images.
- d) Automatically updates the child image as the parent image changes (for example, with security fixes).

Quiz 3

What are two typical scenarios for creating a Dockerfile to build a child image from an existing image? (Choose two.)

- a) Adding new runtime libraries.
- b) Setting constraints to a container's access to the host machine's CPU.
- c) Adding internal libraries to be shared as a single image layer by multiple container images for different applications.
- a) Creates images compatible with OpenShift, unlike container images created from Docker tools.

Quiz 3

What are two typical scenarios for creating a Dockerfile to build a child image from an existing image? (Choose two.)

- a) Adding new runtime libraries.
- b) Setting constraints to a container's access to the host machine's CPU.
- c) Adding internal libraries to be shared as a single image layer by multiple container images for different applications.
- a) Creates images compatible with OpenShift, unlike container images created from Docker tools.



Building Custom Container Images with Dockerfiles

After completing this section, you will be:

Create a container image using

- common Dockerfile commands

Building Images with

- Podman

Building Base Containers

- Dockerfile to automate building of container images
- Three-step process:
 1. Create working directory
 2. Write the Dockerfile
 3. Build the image using `podman build` command
- Working directory consists
 - All files needed to build the image
 - Do not use root directory /
 - Dockerfile is preferred, but changeable
- Write the Dockerfile
 - Text file contains instructions
 - `#` comment
 - INSTRUCTION arguments [arguments ...]

Dockerfile – EXAMPLE

```
# This is a comment line ❶  
FROM ubi7/ubi:7.7 ❷  
LABEL description="This is a custom httpd container image" ❸  
MAINTAINER John Doe <jdoe@xyz.com> ❹  
RUN yum install -y httpd ❺  
EXPOSE 80 ❻  
ENV LogLevel "info" ❼  
ADD http://someserver.com/filename.pdf /var/www/html ❽  
COPY ./src/ /var/www/html/ ❾  
USER apache ❿  
ENTRYPOINT ["/usr/sbin/httpd"] ⓫  
CMD ["-D", "FOREGROUND"] ⓫
```

CMD vs ENTRYPOINT

- Exec form (using JSON array)

```
ENTRYPOINT ["command", "param1", "param2"]  
CMD ["param1", "param2"]
```

- Preferred form.

- Shell form:

```
ENTRYPOINT command param1 param2  
CMD param1 param2
```

- Wraps commands in `/bin/sh -c shell`
- Additional shell process, resource consumption
- Some limitations, may not work as expected

CMD vs ENTRYPOINT

- Workable:

ENTRYPOINT ["ping"]
CMD ["localhost"]
In container, the actual command: `ping localhost`

- Preferred form.

- Not workable:

ENTRYPOINT ["ping"]
CMD localhost
In container, the actual command: `ping /bin/sh -c localhost`

CMD vs ENTRYPOINT

- At most one ENTRYPOINT and one CMD instruction
- Last instruction effective
- Example 1: When container start, it always display current time:

```
ENTRYPOINT ["/bin/date", "+%H:%M"]
```

- Example 2: Combine with CMD, or just CMD alone

```
ENTRYPOINT ["/bin/date"]  
CMD [ "+%H:%M" ]
```

```
CMD ["/bin/date", "+%H:%M"]
```

- When container starts without user parameter, current time is played:

```
[student@workstation ~]$ sudo podman run -it do180/rhel  
11:41
```

- When container starts with user parameter, it overrides the CMD:

```
[student@workstation ~]$ sudo podman run -it do180/rhel +%A  
Tuesday
```

CMD vs ENTRYPOINT

The table below shows what command is executed for different **ENTRYPOINT** / **CMD** combinations:

	No ENTRYPOINT	ENTRYPOINT exec_entry p1_entry	ENTRYPOINT ["exec_entry", "p1_entry"]
No CMD	<i>error, not allowed</i>	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry
CMD ["exec_cmd", "p1_cmd"]	exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry exec_cmd p1_cmd
CMD ["p1_cmd", "p2_cmd"]	p1_cmd p2_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry p1_cmd p2_cmd
CMD exec_cmd p1_cmd	/bin/sh -c exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry /bin/sh -c exec_cmd p1_cmd

ADD and COPY

Have two forms:

- The Shell form:

```
ADD <source> ... <destination>  
COPY <source> ... <destination>
```

- The Exec form:

```
ADD ["<source>", ... "<destination>"]  
COPY ["<source>", ... "<destination>"]
```

- ADD instruction allows specifying resource using URL:

```
ADD http://someserver.com/filename.pdf /var/www/html
```

- If source file is compressed file, ADD decompress.

NOTE: except *.zip files

Layering Image

- Each instruction creates a new image layer
- Multilayers – larger images

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

- Single layer but hard to read

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

- Single layer and much better readability 😊

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

Layering Image

- Applicable to other INSTRUCTIONS

```
LABEL version="2.0" \  
    description="This is an example container image" \  
    creationDate="01-09-2017"
```

```
ENV MYSQL_ROOT_PASSWORD="my_password" \  
    MYSQL_DATABASE "my_database"
```

```
EXPOSE 8080, 9090, 9191
```

Building Images with Podman

- Use of `podman build -t` command
- Example:


```
$ podman build -t NAME:TAG DIR [--file , -f filename ]
```

`-t NAME` : Assign a image name and tag. If `tag` unspecified, default to latest

`DIR` : if Dockerfile is in current working directory, simply use “.”

`--file, -f` : Specifies a different Containerfile which contains instructions for building the image. Can be local file or an http or https URL. Multiple Containerfile can be specified.

Guided Exercise: Creating a Basic Apache Container Image



You should be able to:

- Write a Dockerfile
- Build image using the Dockerfile

Chapter Summary

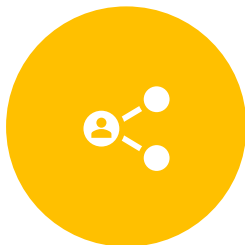
In this chapter, you learned:



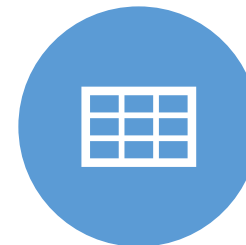
Dockerfile contains instructions that specify how to construct a container image.



The Source-to-Image (S2I) process provides an alternative to Dockerfiles. S2I implements a standardized container image build process for common technologies from application source code. This allows developers to focus on application development and not Dockerfile development.



Container images provided by Red Hat Container Catalog or Quay.io are a good starting point for creating custom images for a specific language or technology.



Building an image from a Dockerfile using a three-step process