BUILDING CONTAINER IMAGES

**Introduction & Learning Objectives**

### BUILDING CONTAINER IMAGES

### Knowledge Check

Congratulations on completing Chapter 4 - Building Container Images. Take this quiz to check your understanding of the concepts you've learned about so far.

BUILDING CONTAINER IMAGES Introduction & Learning Objectives BUILDING CONTAINER IMAGES Knowledge Check Congratulations on completing Chapter 4 - Building Container Images. Take this quiz to check your understanding of the concepts you've learned about so far. Question 4.1 It is possible to create Docker image out of a Docker container. True or False? A. True B. False

ChatGPT

A. True

The statement is True.

It is possible to create a Docker image out of a Docker container. This process is known as "committing" a container. When you commit a container, you are essentially creating a new image from its current state.

You can use the **docker commit** command to commit a container and create a new image from it. The new image will include all the changes made to the container since its initial creation. However, it is important to note that committing containers is generally not recommended for creating production-ready images, as it may lead to inconsistent and undocumented images.

For production use, it is best to define the desired state of a container using a Dockerfile and then use the **docker build** command to build a clean and repeatable image from the Dockerfile. This ensures that the image creation process is standardized and well-documented, making it easier to manage and maintain container images in the long run.



Question 4.2 Docker Hub provides private registry service. True or False? A. True B. False

ChatGPT

A. True

The statement is True.

Docker Hub provides both public and private registry services. Docker Hub is a cloud-based registry service provided by Docker, Inc., where users can store, share, and manage Docker container images.

Public registry: Docker Hub allows users to store and share Docker images publicly, meaning anyone can access and pull the images from the Docker Hub repository.

Private registry: Docker Hub also offers private registry services. With a private registry, users can store Docker images privately, restricting access to specific teams or collaborators. This is useful for organizations that want to keep their container images private and control who can access and use them.

The private registry feature in Docker Hub allows teams to manage and distribute container images within their organization securely. It provides additional control and security over the container images, ensuring that sensitive or proprietary images are not publicly accessible.



Question 4.3 Which of the following is the recommended way of creating Docker images?

ChatGPT

The recommended way of creating Docker images is by using Dockerfiles.

Dockerfiles are simple text files that contain a set of instructions for building a Docker image. They provide a declarative and version-controlled approach to define the desired state of a containerized application or service. Dockerfiles include commands to install dependencies, copy files, set environment variables, and configure the application inside the container.

Using Dockerfiles has several advantages:

1. **Reproducibility:** Dockerfiles provide a standardized way to define the image's configuration, making it easy to reproduce the same image on different systems and environments.
2. **Version control:** Dockerfiles can be stored in version control systems (e.g., Git), allowing teams to collaborate and track changes to the image definition over time.
3. **Transparency:** Dockerfiles are human-readable and transparent, making it clear what goes into the image and how it is built.
4. **Automation:** Dockerfiles can be used with CI/CD pipelines to automate the image building process, ensuring consistent and reliable image creation.
5. **Layered approach:** Docker uses a layered file system for images, and Dockerfiles enable you to leverage this layered approach to optimize image builds and minimize duplication.

To build a Docker image using a Dockerfile, you use the **docker build** command, specifying the path to the directory containing the Dockerfile. Docker then reads the Dockerfile instructions and creates the image based on those instructions.

Overall, Dockerfiles are the recommended way of creating Docker images due to their simplicity, maintainability, and compatibility with version control and automation tools.



Question 4.4 What does the docker commit command do? A. It pushes the Docker image to Docker Hub B. It converts an image into a container C. It creates an image from a container's changes D. It pulls the image from a Docker Hub

ChatGPT

C. It creates an image from a container's changes.

Option C is correct. The **docker commit** command is used to create a new Docker image from the changes made to a running container.

When you commit a container using **docker commit**, it takes a snapshot of the container's file system and other settings, creating a new image with those changes. The new image will include all the modifications made to the container since it was started, effectively capturing the current state of the container in a new image.

For example, if you have a running container with an application and you've made some customizations or installed additional packages inside the container, you can commit those changes to create a new image. The new image will include the changes you made, and you can use it to create new containers with the same modifications.

However, it is important to note that committing containers is generally not recommended for creating production-ready images. This approach can lead to inconsistent and undocumented images, making it harder to manage and maintain container images in the long run. For production use, it is best to define the desired state of a container using a Dockerfile and use the **docker build** command to create a clean and repeatable image from the Dockerfile. This ensures consistency and standardization in the image creation process.



Question 4.5 When you use COPY . . the files will be copied to a \_\_\_\_\_\_\_\_\_ [destination]. Fill in the blank. A. Container B. Build context C. Docker daemon D. Working directory (WORKDIR)

ChatGPT

A. Container

When you use the **COPY . .** command in a Dockerfile, the files from the build context (the current directory on the host where the Dockerfile is located) will be copied to a specified destination inside the container. The first dot (**.**) represents the source directory in the build context, and the second dot represents the destination directory inside the container.

For example, if you have the following line in your Dockerfile:

bashCopy code

COPY . /app

It means that all the files and directories in the current directory (build context) on the host will be copied to the **/app** directory inside the container when the image is built.

By using **COPY** in the Dockerfile, you can include files from the host machine into the image during the build process. This allows you to package your application code, configuration files, and other assets into the image, making them available to the container when it runs.



Question 4.6 When you have both ENTRYPOINT and CMD defined, which of the following will be called first after launching the container? A. ENTRYPOINT B. CMD C. RUN D. LAUNCH

ChatGPT

A. ENTRYPOINT

When both ENTRYPOINT and CMD are defined in a Dockerfile, the ENTRYPOINT instruction takes precedence and is executed first after launching the container.

The **ENTRYPOINT** instruction in a Dockerfile sets the default application or command that will be executed when the container starts. It defines the primary executable for the container, and any arguments provided when running the container are treated as arguments to the entrypoint command.

The **CMD** instruction, on the other hand, is used to provide default arguments to the **ENTRYPOINT** command. If there are no arguments provided when running the container, the **CMD** values will be used as the default.

Here's an example Dockerfile:

lessCopy code

FROM ubuntu:latest ENTRYPOINT ["echo", "Hello"] CMD ["World"]

In this example, if you run the container without specifying any additional command-line arguments, it will print "Hello World" because the ENTRYPOINT command is executed first, and the CMD value "World" is used as the default argument.

If you run the container with additional arguments, they will be appended to the ENTRYPOINT command. For instance:

arduinoCopy code

docker run my\_image\_name John

This will print "Hello John" because "John" is provided as an additional argument to the **echo** command specified in the **ENTRYPOINT**.



Question 4.7 If you define WORKDIR as /xyz/pqr/abc, and if /xyz/pqr (parent directory) does not exist, image build will fail while creating working directory. True or False? A. True B. False

ChatGPT

A. True

The statement is True.

If you define **WORKDIR** as **/xyz/pqr/abc** in your Dockerfile, and the parent directory **/xyz/pqr** does not exist, the image build will fail when Docker tries to create the working directory.

When the Docker image is built, each instruction in the Dockerfile is executed sequentially. When it encounters the **WORKDIR** instruction, Docker will try to create the specified working directory. However, if any parent directories in the path do not exist, Docker will not automatically create them, and the build process will fail with an error.

To avoid this issue, you should ensure that the parent directories of the **WORKDIR** are created before specifying it in the Dockerfile. One way to do this is by creating the necessary parent directories in a previous step of the Dockerfile or by using the **RUN** instruction to create the required directories.

For example:

bashCopy code

FROM ubuntu:latest # Create the parent directories /xyz/pqr RUN mkdir -p /xyz/pqr # Set the working directory to /xyz/pqr/abc WORKDIR /xyz/pqr/abc # Rest of the Dockerfile...

By including the **RUN mkdir -p /xyz/pqr** instruction before setting the **WORKDIR**, you ensure that the parent directory exists, and the **WORKDIR** instruction will not cause the build to fail.



Question 4.8 Which of the following have an impact on the size of the container image? A. Redundant RUN instructions B. Base image defined with FROM C. Files copied using COPY D. All of the above

ChatGPT

D. All of the above

All the options mentioned have an impact on the size of the container image:

A. Redundant RUN instructions: Each **RUN** instruction in a Dockerfile creates a new layer in the container image. If there are redundant or unnecessary **RUN** instructions in the Dockerfile, it can result in larger image size due to unnecessary layer creations.

B. Base image defined with FROM: The base image specified in the **FROM** instruction affects the starting point of the image. Larger base images can lead to larger overall image sizes.

C. Files copied using COPY: When you use the **COPY** instruction to include files from the host into the image, those files become part of the image's filesystem, increasing the image size.

Layering is a fundamental concept in Docker images. Each instruction in a Dockerfile creates a new layer, and each layer represents a set of filesystem changes. Docker uses a union file system to combine these layers into the final image. When multiple layers contain similar or redundant data, it can result in a larger image size.

To optimize the image size, it is essential to carefully design the Dockerfile, minimize the number of layers, use efficient base images, and avoid unnecessary or redundant instructions. Additionally, using multi-stage builds, where intermediate build artifacts are discarded in the final image, can also help reduce the image size.

# Question 4.1

It is possible to create Docker image out of a Docker container. True or False?

Correct Answer

* **A.** True

Your Answer:

Correct

* **B.** False

# Question 4.2

Docker Hub provides private registry service. True or False?

Correct Answer

* **A.** True

Your Answer:

Correct

* **B.** False

# Question 4.3

Which of the following is the recommended way of creating Docker images?

Correct Answer

* **A.** Modifying a container and committing it
* **B.** By writing a Dockerfile

Your Answer:

Correct

* **C.** Using docker-compose files
* **D.** None of the above

# Question 4.4

What does the **docker commit** command do?

Correct Answer

* **A.** It pushes the Docker image to Docker Hub
* **B.** It converts an image into a container
* **C.** It creates an image from a container's changes

Your Answer:

Correct

* **D.** It pulls the image from a Docker Hub

# Question 4.5

When you use **COPY . .** the files will be copied to a \_\_\_\_\_\_\_\_\_ [destination]. Fill in the blank.

Incorrect Answer

* **A.** Container

Your Answer:

Incorrect

* **B.** Build context
* **C.** Docker daemon
* **D.** Working directory (WORKDIR)

Correct Answer

# Question 4.6

When you have both ENTRYPOINT and CMD defined, which of the following will be called first after launching the container?

Correct Answer

* **A.** ENTRYPOINT

Your Answer:

Correct

* **B.** CMD
* **C.** RUN
* **D.** LAUNCH

# Question 4.7

If you define WORKDIR as **/xyz/pqr/abc**, and if **/xyz/pqr** (parent directory) does not exist, image build will fail while creating working directory. True or False?

Incorrect Answer

* **A.** True

Your Answer:

Incorrect

* **B.** False

Correct Answer

# Question 4.8

Which of the following have an impact on the size of the container image?

Correct Answer

* **A.** Redundant RUN instructions
* **B.** Base image defined with FROM
* **C.** Files copied using COPY
* **D.** All of the above

Your Answer:

Correct