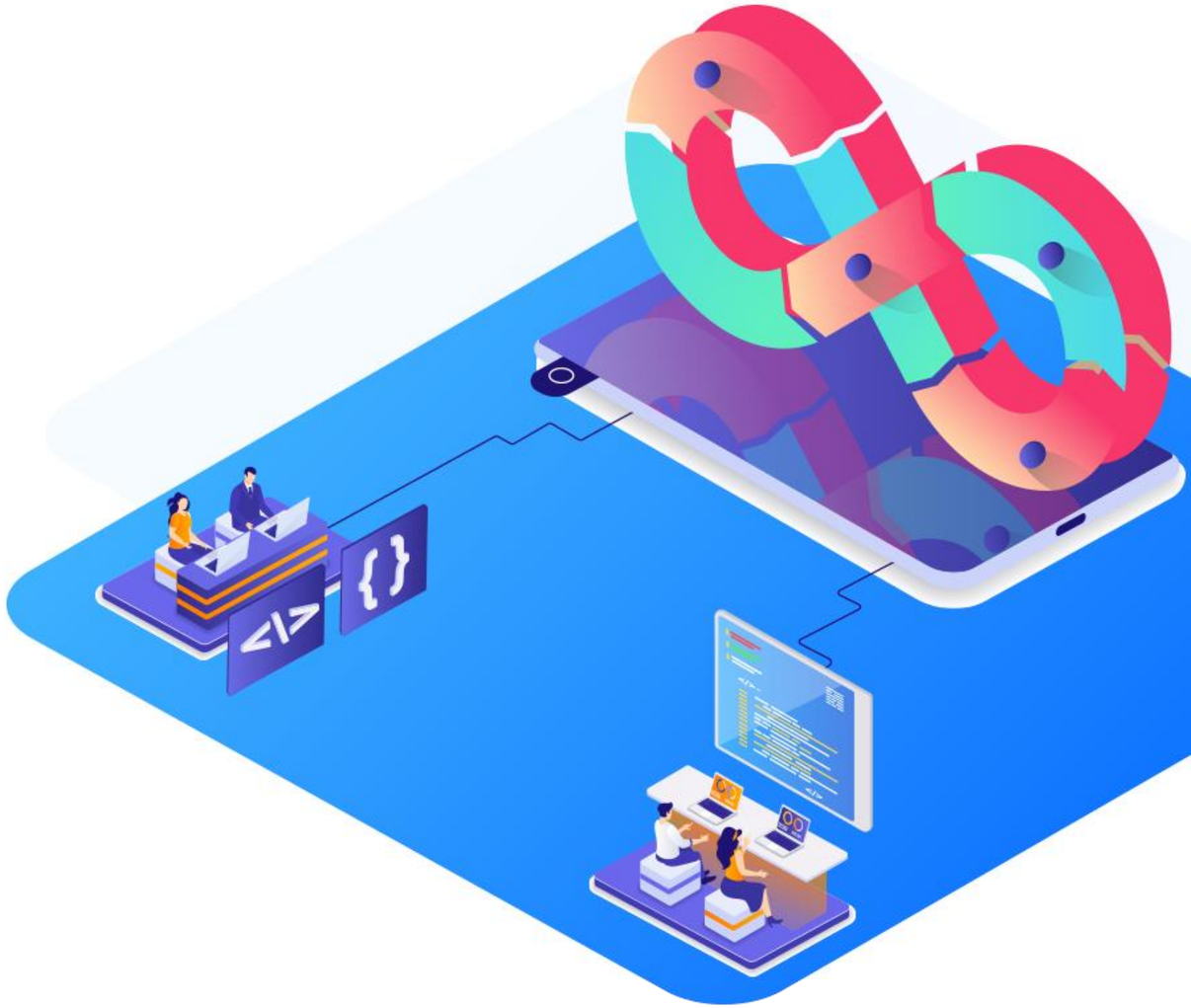
Containerization with Docker



Managing Docker Images and Registries

Learning Objectives

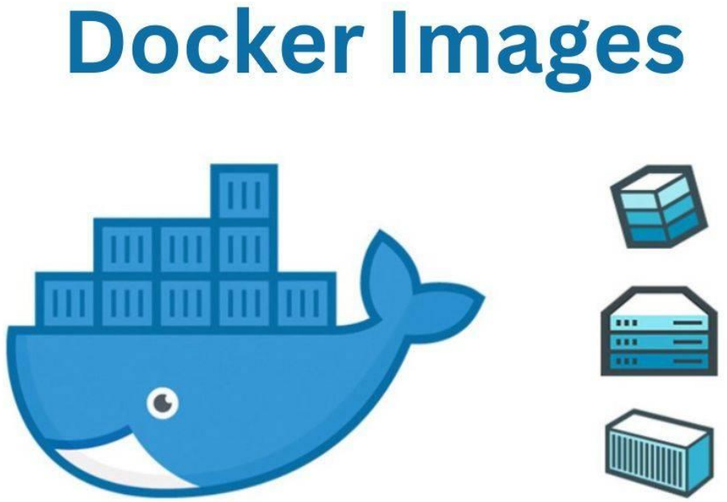
By the end of this lesson, you will be able to:

- Build Docker images using Dockerfiles, emphasizing the manipulation and optimization of image layers
- Outline the container lifecycle to allow optimization of resource allocation
- Use Docker commands to pull, push, tag, and remove images, including managing images across different environments and registries
- Set up and configure both public and private Docker registries, deploy images to these registries, and manage access and security settings

Introduction to Docker Images and Containers

Docker Image: Overview

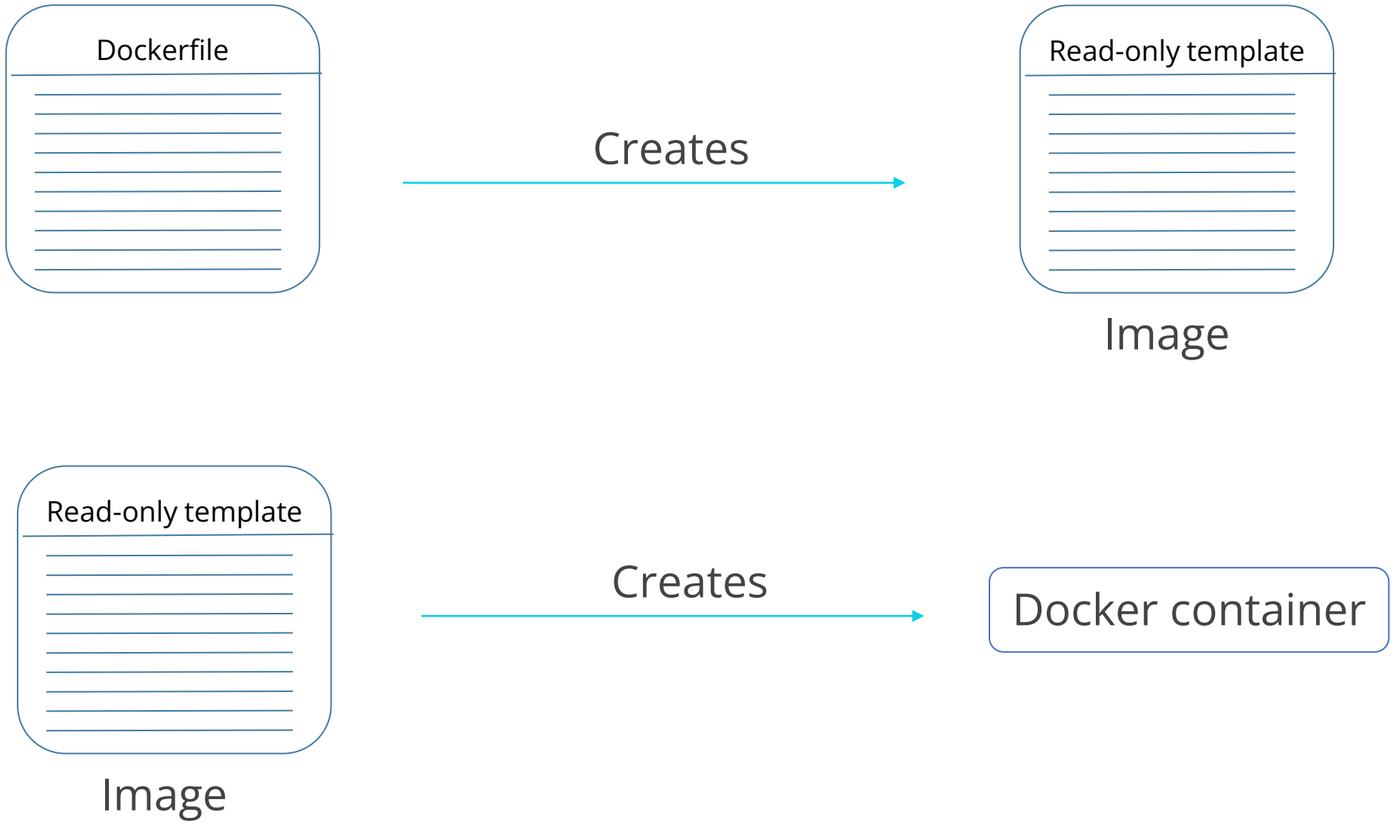
A Docker image is a file that runs programs within a Docker container. It functions as a set of guidelines like a template.



Docker images also behave as the starting point when using Docker. An image is similar to a snapshot in virtual machine (VM) environments.

Docker Image: Overview

An image holds instructions that are required to run an application.



Parts of a Docker Image

Base image

The user can build this first layer entirely from scratch with the build command.

Parent image

This can be the first layer in a Docker image which is a reused image that serves as a foundation for all other layers.

Layers

Layers are added to the base image, using code that will enable it to run in a container.

Parts of a Docker Image

Container layer

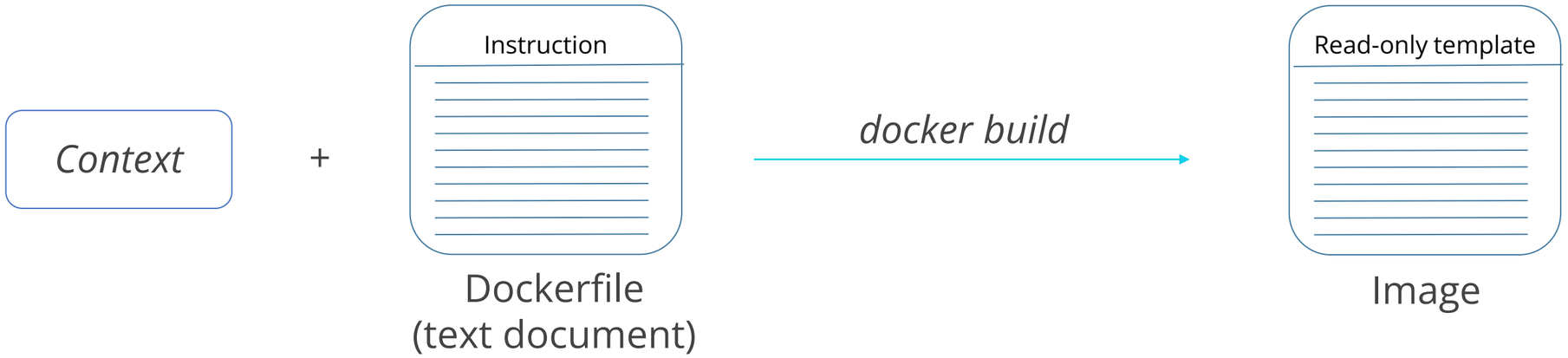
A Docker image not only creates a new container, but also a writable or container layer which is used to customize the containers.

Docker manifest

This part of the Docker image is an additional file which uses JSON format to describe the image.

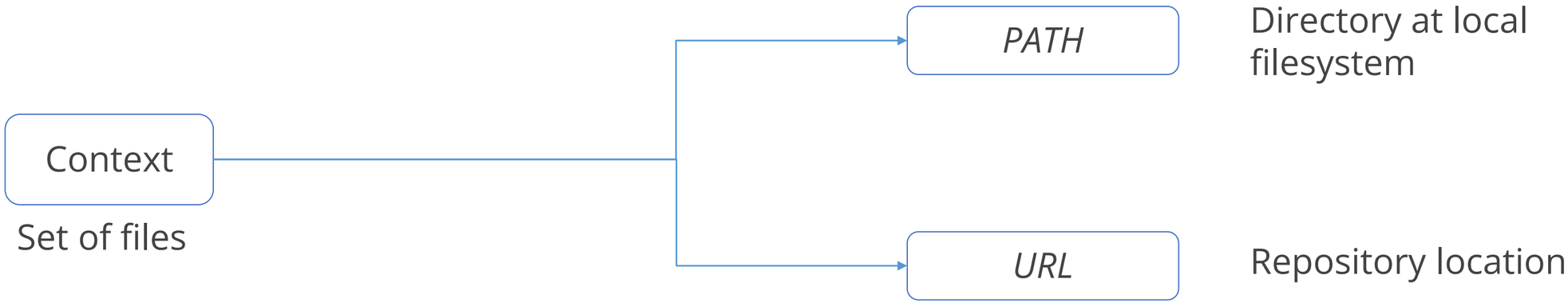
Dockerfile: Overview

It contains all the necessary instructions that are used to build images.



The *docker build* command creates an image from a *context* and a *Dockerfile*.

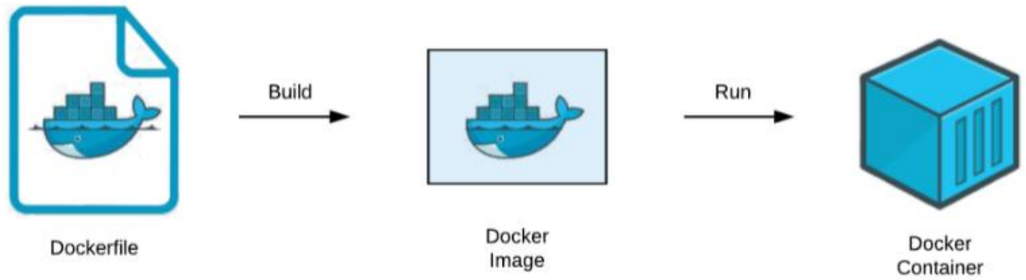
Dockerfile: Overview



Use of current directory in the *docker build* command → *$ docker build .*

Docker Containers: Overview

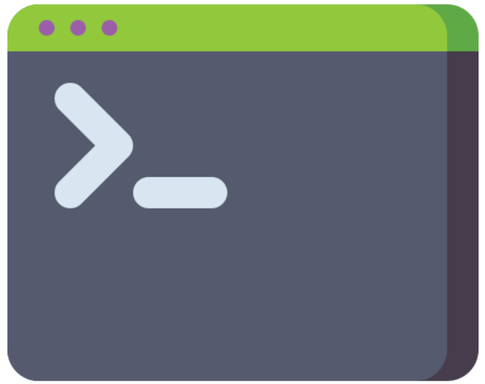
A container is a standardized software component that wraps up code and its dependencies to ensure that an application runs consistently in different computing environments.



An executable software package known as a Docker container image contains all the components required to run a program, including the code, runtime, system tools, system libraries, and settings.

Docker Containers: Execution

A container is the same as a typical operating system process, with the exception that it is isolated and has its own file system, networking, and isolated process tree that is distinct from the host.

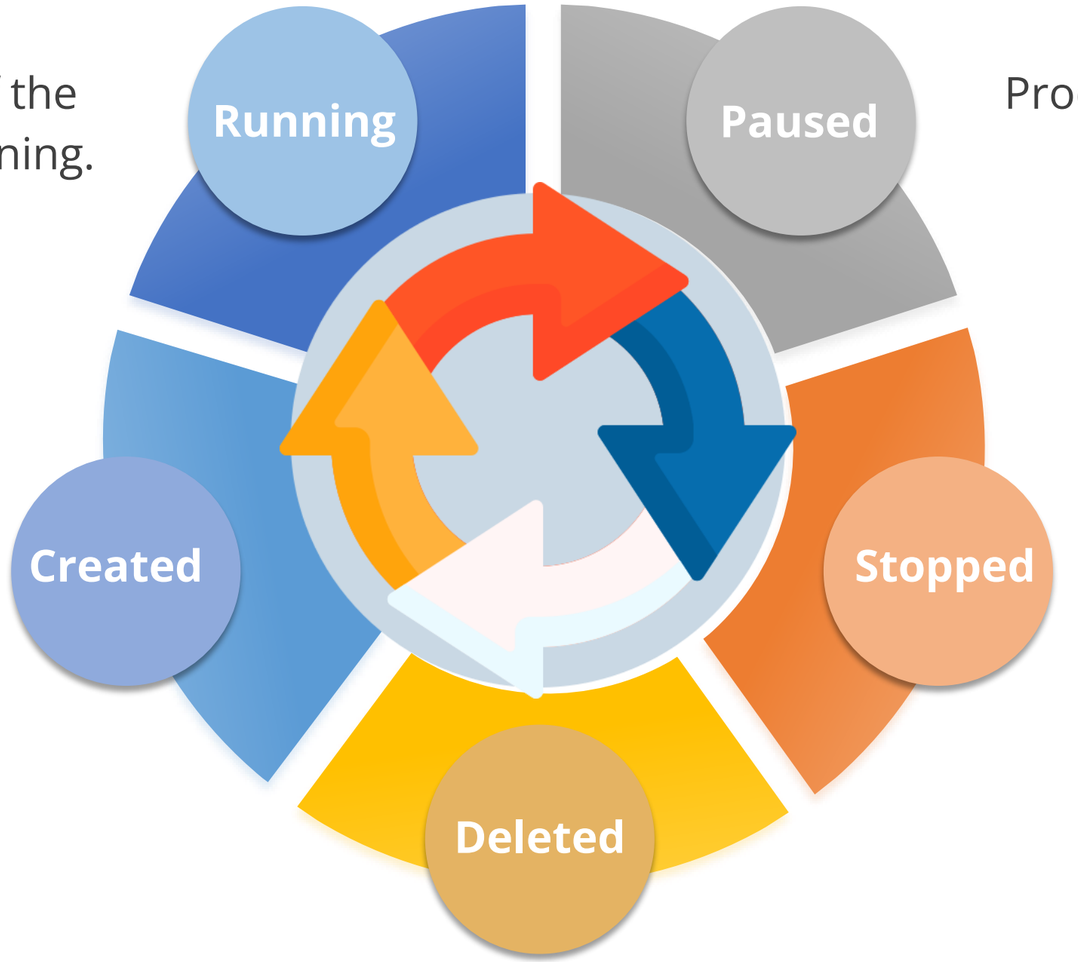


The **docker run** command is used to execute an image inside of a Docker container. The image name is the only parameter needed for the **docker run** command.

Container Lifecycle

Here are the main phases in the lifecycle of a Docker container:

The processes of the container are running.



Processes have been paused for some time.

The container is created but not started.

The process has been stopped.

The container is deleted, including all its processes.

Container Lifecycle

To create the new Docker container with an image, use the given command:

```
$ docker create  --name <container-name> <image-name>
```

Container Lifecycle

To start a newly created or stopped Docker container, use the **docker start** command.

```
$ docker start  <container-name>
```

Example:

```
$ docker start docker-container-2022
```

Container Lifecycle

The **docker run** command performs the tasks of both the **docker create** and **docker start** commands.

```
$ docker run -it --name  <container-name>
```

Example:

```
$ docker run -it --name docker-container-2022
```

Container Lifecycle

The **docker pause** command is used to pause a running container by specifying its name.

```
$ docker pause  <container-name>
```

Example:

```
$ docker pause docker-container-2022
```

Container Lifecycle

Syntax:

```
$ docker stop  <container-name>
```

Example:

```
$ docker stop docker-container-2022
```

Container Lifecycle

To delete all the containers, use the given command:

```
$ docker rm $(docker container ps -a)
```

Example:

```
$ docker rm $(docker container ps -a)
a2f73432da1c
dda9c86d423e
3b326aa26db6
2668cf8c9aab
```

Container Lifecycle

When using the **docker stop** command, all processes running in the designated container are stopped.

A stopped container can be transitioned into a running state.

Container Lifecycle

Before deleting a Docker container, it is necessary to stop all running processes inside it.

It is recommended to pause the container before making any modifications to avoid encountering errors.

Assisted Practice

Demonstrating lifecycle of containers **Duration: 10 Min.**

Problem statement:

You have been assigned a task to demonstrate the lifecycle of containers for efficient management and optimization of Docker container orchestration.

Outcome:

By completing this demo, you will be able to effectively manage the lifecycle of Docker containers, ensuring efficient container orchestration and optimization.

Note: Refer to the demo document for detailed steps
01_Demonstrating_Lifecycle_of_Containers

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Demonstrate Docker container lifecycle management

ASSISTED PRACTICE

Assisted Practice

Creating a Docker Image **Duration: 10 Min.**

Problem statement:

You have been assigned a task to create a Docker image using a Docker file, which installs and configures a Nginx web server with a custom welcome message.

Outcome:

By completing this demo, you will be able to create a Docker image containing a Nginx web server with a custom welcome message using a Dockerfile. This image can then be used to spin up containers with the configured web server environment.

Note: Refer to the demo document for detailed steps
02_Creating_a_Docker_Image

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Create a Docker image using the Docker file

ASSISTED PRACTICE

Quick Check

You are working on a DevOps project and want to debug specific application components. Which command will you use to pause a running container by specifying its name?

A. $ docker hold execution myContain

B. $ docker pause myContain

C. $ docker break –it --name myContain

D. $ docker kill myContain

Image Management

Why Is It Necessary?

Efficient resource management

Proper image management minimizes storage space by avoiding redundant images, leading to more efficient resource use and cost savings

Consistency and reliability

Managing images ensures that applications run consistently across different environments, reducing deployment issues and improving reliability

Security and compliance

Effective image management helps maintain security by using up-to-date images and scanning for vulnerabilities, minimizing the risk of breaches

Scalability and automation

Enable faster deployments and smooth scalability in automated environments like Kubernetes, ensuring seamless operations at scale

Image Creation Techniques

Interactive method

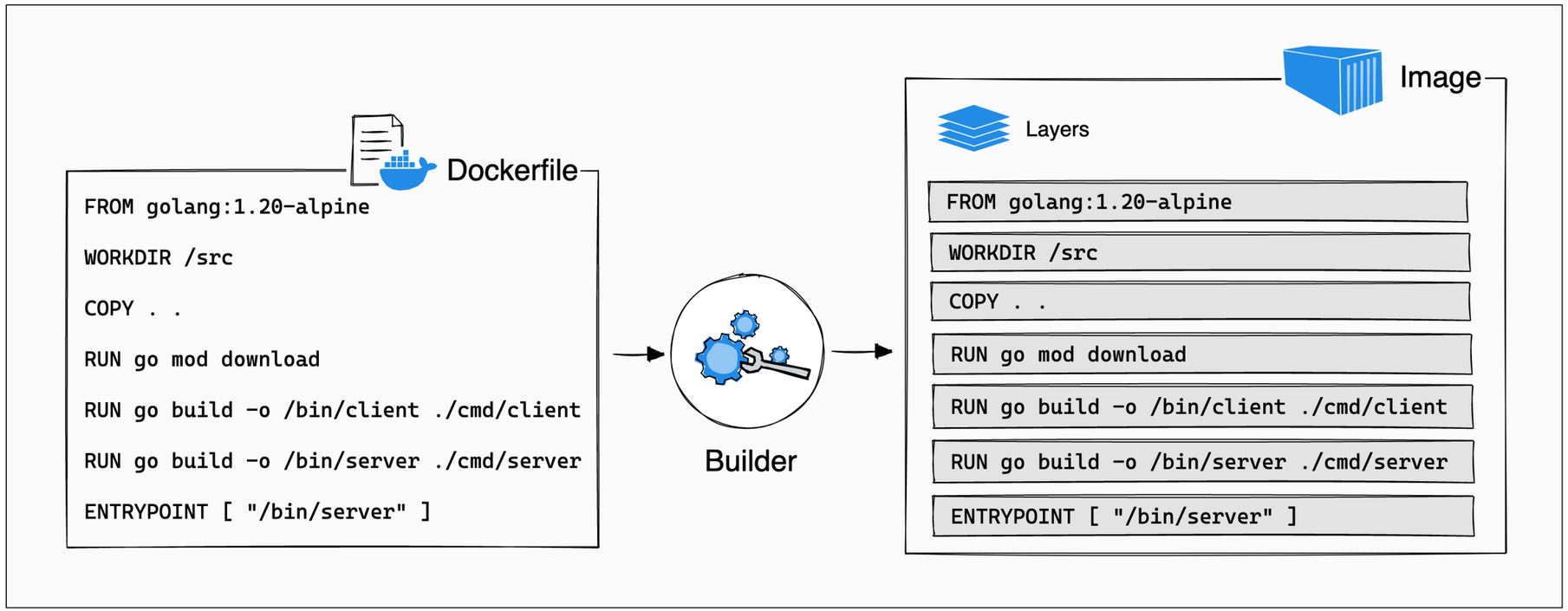
- This is the easiest way to create Docker images.
- With this method, users run a container from an existing Docker image and manually make any needed changes to the environment before saving the image.
- **docker build -t <image_name>:<tag>** is the command to create a docker image through CLI.

Dockerfile method

- This process is more difficult and time-consuming, but it does well in continuous delivery environments.
- This approach requires making a plain text Dockerfile. The Dockerfile makes the specifications for creating an image.
- Dockerfile contains all necessary dependencies defined in it to build a docker image

Layers of Docker Image

A Docker image is composed of multiple layers stacked on top of each other. The following diagram illustrates how a Dockerfile translates into a stack of layers in a container image:



Source: https://docs.docker.com/build/guide/layers/

Layers of Docker Image

Building layers consists of the following four major components:

Dockerfile	It is a file that contains the commands or instructions to build layers.
Imagedb	It stores images in registries like Docker Hub.

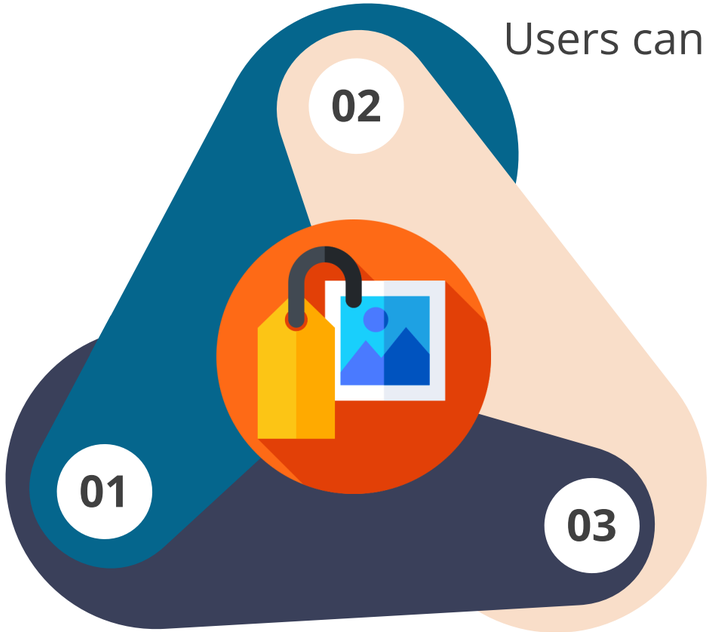
Layers of Docker Image

Layerdb	It stores the current working state of layers as they are built at each command from Dockerfile.
Caches	It stores objects or files for images so they can be cached to speed up retrieval time.

How to Tag an Image?

02

Users can tag Docker images based on their preferences.



01

Identifying a specific image becomes difficult when there are multiple images.

03

An image name comprises of name components separated by slashes.

Tagging an Image

- **Tag by ID**
- **Tag by name**
- **Tag by name and tag**
- **Tag for a private repository**

Users employ it to tag a local image with ID *0e93876876* and associate it with the spring repository using version 1.0.

```
docker tag 0e93876876 spring/myapp:version1.0
```

Tagging an Image

- Tag by ID
- **Tag by name**
- Tag by name and tag
- Tag for a private repository

It is used to tag a local image with the name *myapp* into the spring repository with version1.0:

```
docker tag myapp spring/myapp:version1.0
```

Tagging an Image

- Tag by ID
- Tag by name
- **Tag by name and tag**
- Tag for a private repository

It is used to tag a local image with the name *myapp* and tag *test* into the spring repository with version1.0.test:

```
docker tag httpd:test spring/myapp:version1.0.test
```

Tagging an Image

- Tag by ID
- Tag by name
- Tag by name and tag
- Tag for a private repository

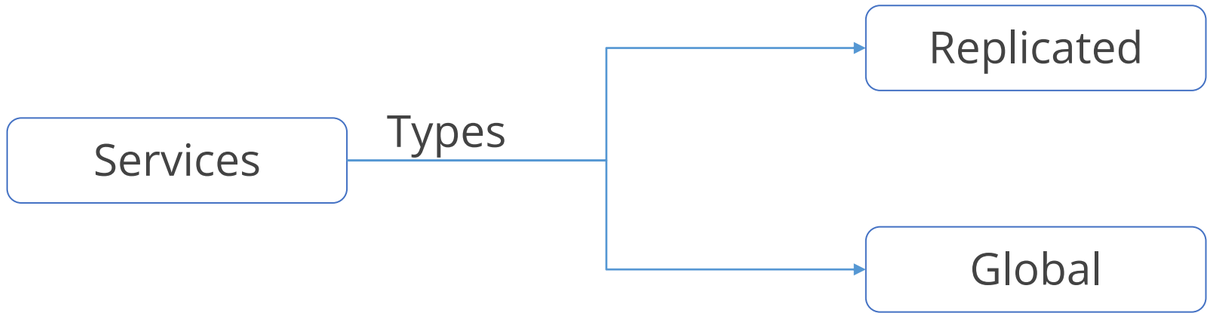
It facilitates pushing an image to a private registry rather than the central Docker registry.

```
docker tag 0e93876876
myregistryhost:9090/spring/myapp:version1.0
```

Services and Tasks: Overview

Services are part of the image that perform specific tasks, allowing you to scale containers across multiple Docker Daemons. These Daemons work together in a network with various manager and worker nodes.

Each member of a network is a Docker Daemon that communicates with other Daemons using the Docker API.



A service allows you to define the desired state, such as the number of replicas of the service that must be available at any given time. By default, the service is load-balanced across all worker nodes.

Docker Commit

This command commits a container's changes or settings to a new image. By default, all processes are paused during the commit process, which helps reduce data corruption.

Committing a container:

```
$ docker ps

$ docker commit ContainerID repository:tag

$ docker images
```

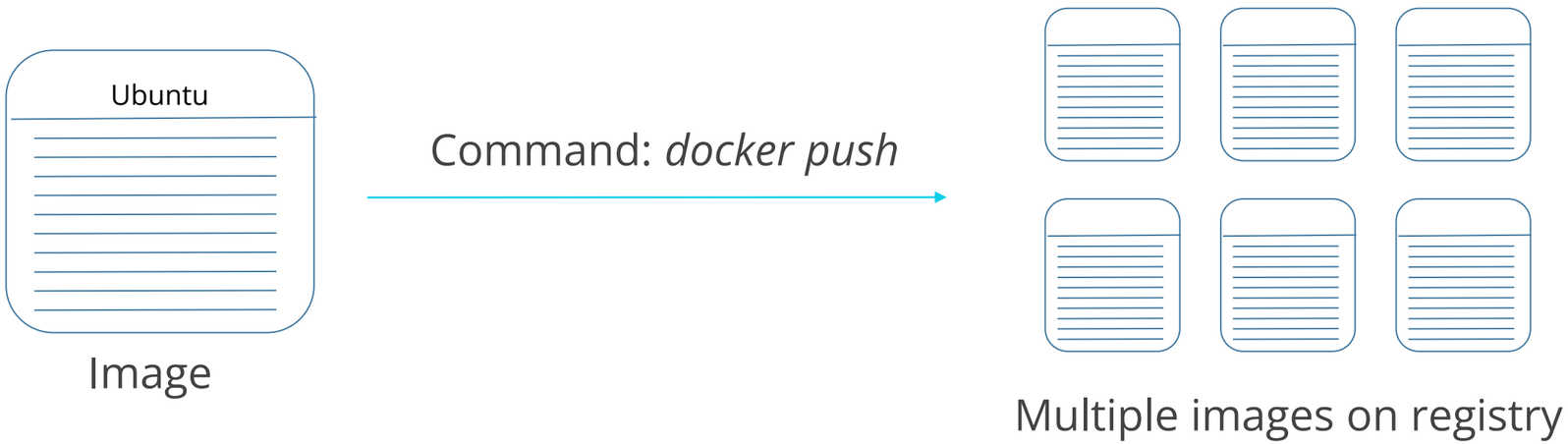
While committing, the **--change** option is used to make changes in Dockerfile instructions such as CMD, ENTRYPOINT, ENV, EXPOSE, LABEL, ONBUILD, USER, VOLUME, and WORKDIR.

Docker Push Command

The **docker push** command is used to upload or push Docker images from a local environment to a Docker registry, such as Docker Hub, Google Container Registry, Amazon ECR, or a private registry.

Syntax:

```
docker push [OPTIONS] NAME[:TAG]
```

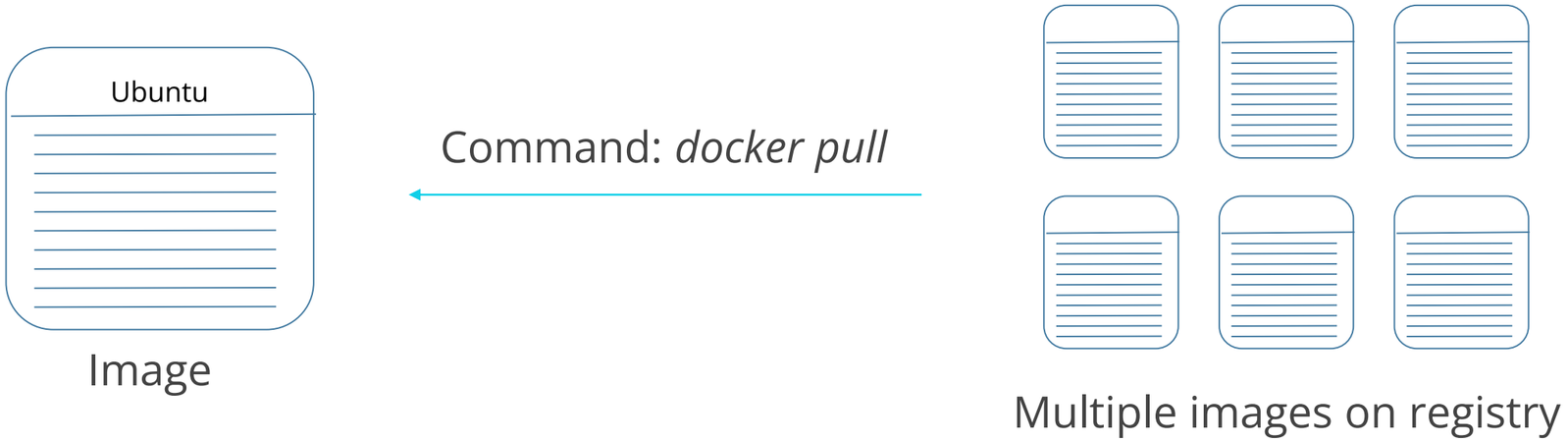


Docker Pull Command

The **docker pull** command downloads Docker images from a Docker registry to the local machine. It is generally needed to set up environments using the pre-built images.

Syntax:

```
docker pull [OPTIONS] NAME[:TAG]
```



Assisted Practice

Tagging an image **Duration: 10 Min.**

Problem statement:

You have been assigned a task to demonstrate the process of tagging Docker images using various methods for easier management and distribution.

Outcome:

By completing this demo, you will be able to tag Docker images with different versions and repository names, facilitating easier management and distribution of Docker images.

Note: Refer to the demo document for detailed steps
03_Tagging_an_Image

Assisted Practice: Guidelines

Steps to be followed:

1. Pull a Docker image
2. Tag an image referenced by name
3. Tag an image for a private repository

ASSISTED PRACTICE

Assisted Practice

Displaying layers of a Docker image **Duration: 10 Min.**

Problem statement:

You have been assigned a task to display the layered structure of Docker images for providing insight into the hierarchical arrangement of image layers within Docker's architecture.

Outcome:

By completing this demo, you will be able to create a Docker image with multiple layers and display the layered structure, providing insight into the hierarchical arrangement of image layers within Docker's architecture.

Note: Refer to the demo document for detailed steps
04_Displaying_Layers_of_a_Docker_Image

Assisted Practice: Guidelines

Steps to be followed:

1. Create and display the layers of a Docker image

Quick Check

As a DevOps engineer, which image layers will you modify to store the current working state of layers as they are built at each command from Dockerfile?

A. Layerdb

B. Caches

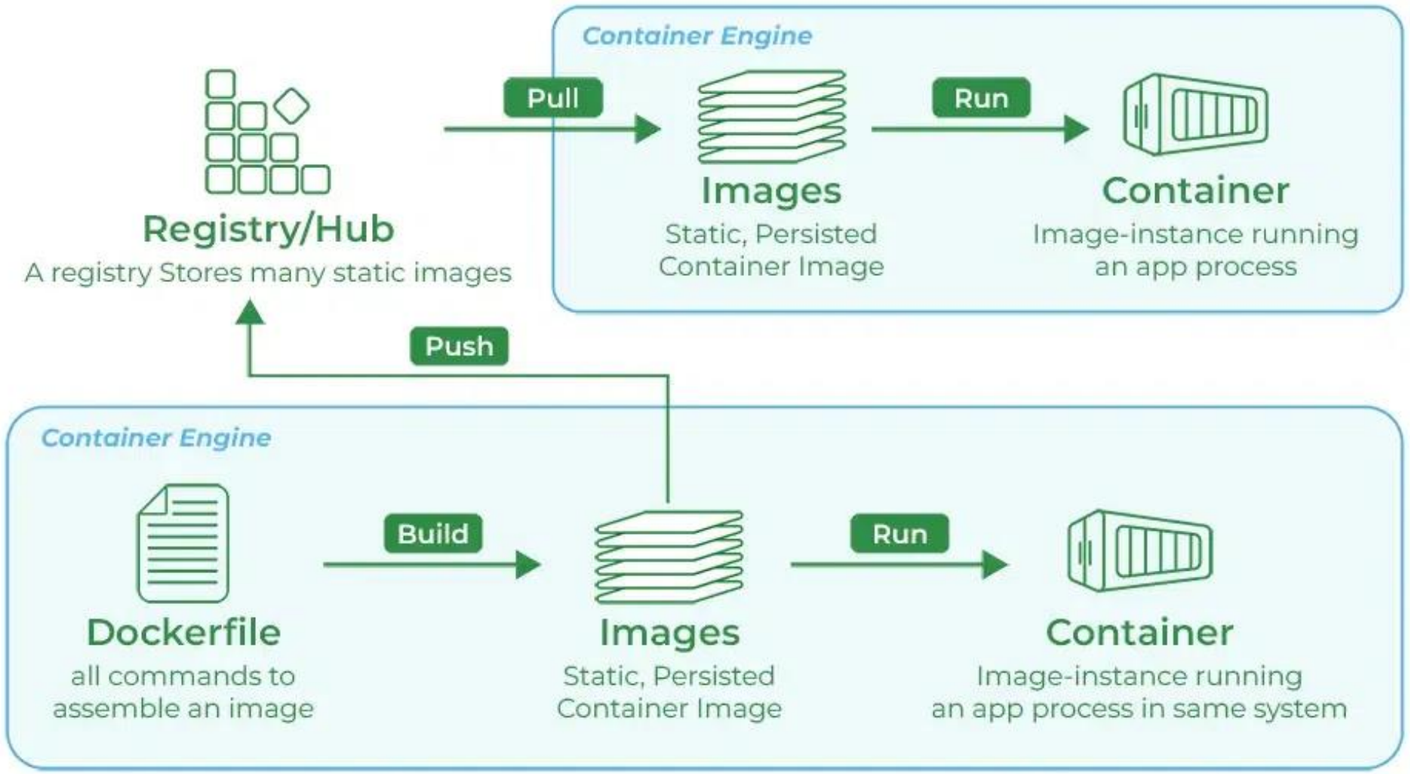
C. Imagedb

D. Dockerfile

Introduction to Docker Registry

Docker Registry: Overview

A Docker registry is a system for storing and distributing Docker images with specific names. It is separated into Docker repositories, each holding all image modifications.



Source: https://www.geeksforgeeks.org/what-is-docker-registry/

Docker Registry: Features

01 Docker registries provide a centralized platform for storing and distributing container images efficiently.

02 Users can maintain multiple versions of the same image within the registry, allowing for version control and image management.

03 Users can set up private registries for storing sensitive images securely, enabling controlled access within an organization.

Docker Registry: Commands

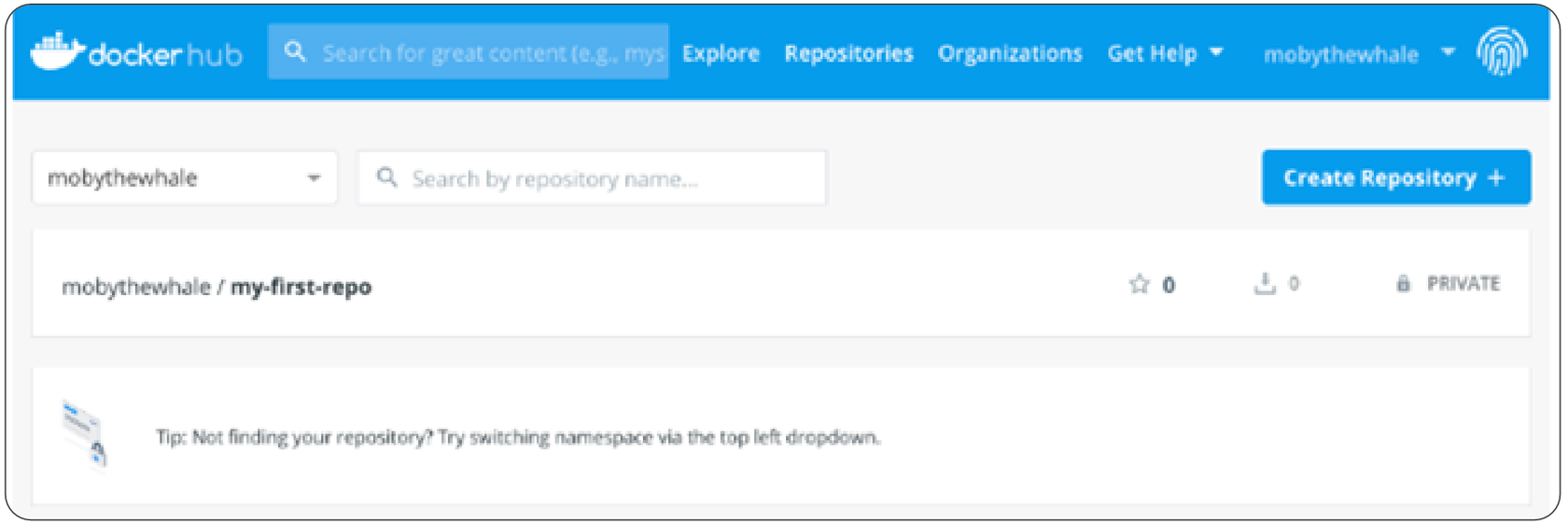
Description	Commands
Starting your registry	docker run -d -p 5000:5000 --restart=always --name registry registry:2
Pulling images from the hub	docker pull ubuntu:latest
Tagging an image and point to registry	docker image tag ubuntu:latest localhost:5000/gfg-image
Pushing the image	docker push localhost:5000/gfg-image
Pulling the image back	docker pull localhost:5000/gfg-image
Stop the registry	docker container stop registry

Docker Hub Repositories

Docker Hub repositories allow the user to share container images with the team, customers, or the Docker community at large.

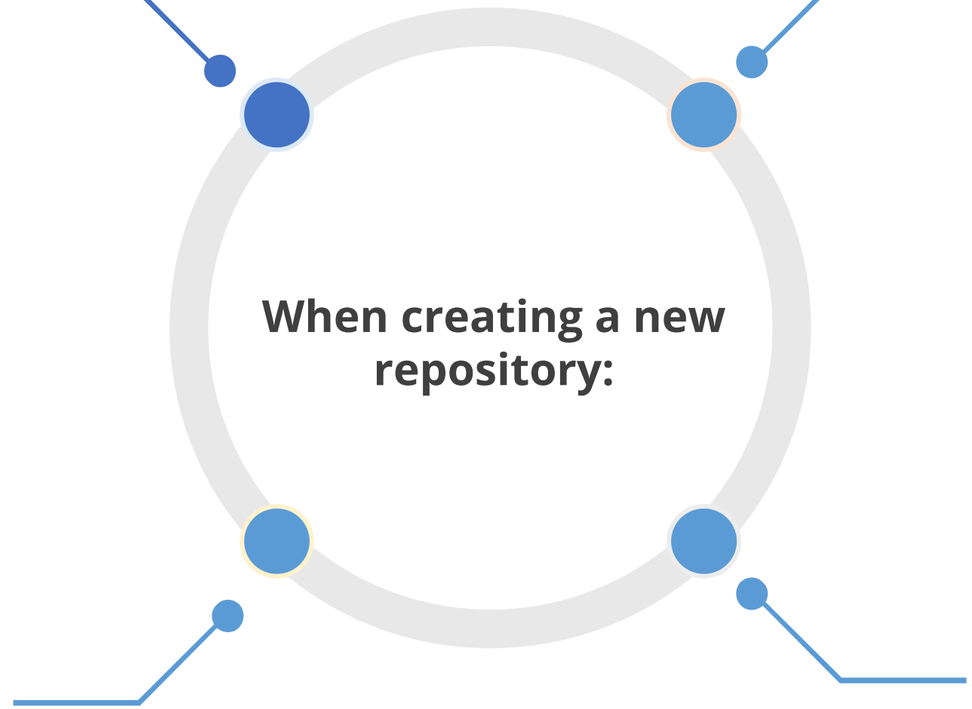
Creating Repositories

1. Sign in to Docker Hub.
2. Click on Create Repository to create a repository.



Source: https://docs.docker.com/docker-hub/repos/

Docker Hub Repositories



The user can choose to put it in their Docker ID namespace

The repository name must be unique in that namespace, can be two to 255 characters, and can only contain lowercase letters, numbers, or **-** and **_**

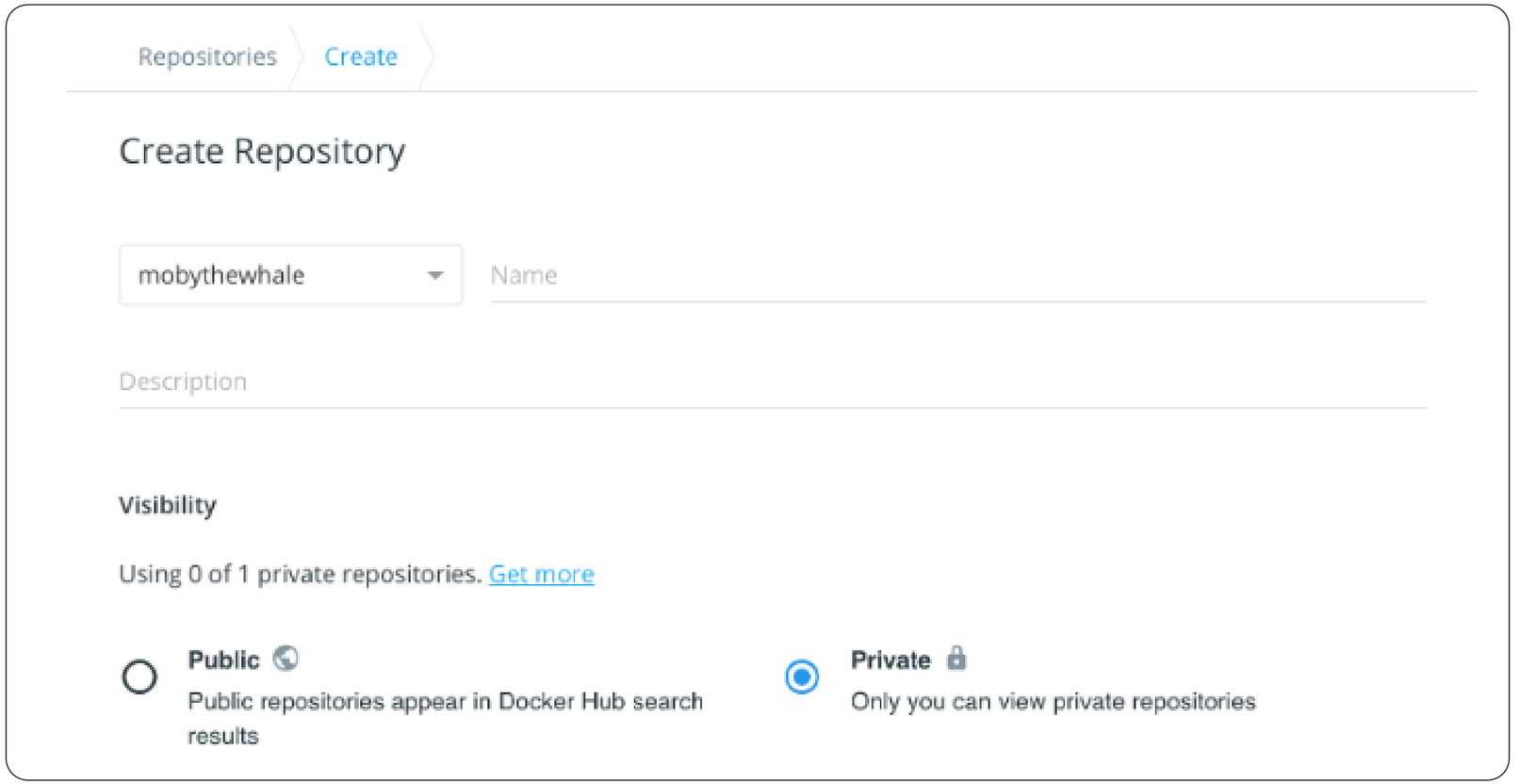
The description can be up to 100 characters and is used in the search result.

The user can link a GitHub or Bitbucket account, or choose to do it later in the repository settings

Private Repositories

Private repositories allow the user to keep container images private, either in their own account or within an organization or team.

To create a private repository, select **Private** when creating a repository:



Source: https://docs.docker.com/docker-hub/repos/

Assisted Practice

Deploying and Configuring a Registry **Duration: 10 Min.**

Problem statement:

You have been assigned a task to configure a local Docker registry for efficient storage and transfer of container images, facilitating streamlined image management within the development environment.

Outcome:

By completing this demo, you will be able to deploy and configure a local Docker registry, including pushing, pulling, and managing Docker images within the registry.

Note: Refer to the demo document for detailed steps
05_Deploying_and_Configuring_a_Registry

Assisted Practice: Guidelines

Steps to be followed:

1. Run a local registry

Assisted Practice

Pushing an Image to Registry **Duration: 10 Min.**

Problem statement:

You have been assigned a task to copy an image from Docker Hub to your local Docker registry.

Outcome:

By completing this demo, you will be able to copy the Nginx image from Docker Hub to your local registry, demonstrating the process of transferring images for efficient local image management.

Note: Refer to the demo document for detailed steps
06_Pushing_an_Image_to_Registry

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Copy an image from the docker hub to your registry

ASSISTED PRACTICE

Assisted Practice

Pulling and Deleting Images from Registry **Duration: 10 Min.**

Problem statement:

You have been assigned a task to pull and delete Docker images from a local Docker registry.

Outcome:

By completing this demo, you will be able to pull images from Docker Hub, tag them for your local registry, push them to the registry, and delete all Docker images from your system.

Note: Refer to the demo document for detailed steps
07_Pulling_and_Deleting_Images_from_Registry

Assisted Practice: Guidelines

Steps to be followed:

1. Pull and delete image in Docker

ASSISTED PRACTICE

Quick Check

You are tasked with pulling a specific version of an image from a private Docker registry hosted at myregistry.local:5000. The image is named myapp, and you need the version tagged as 3.1.4. Which of the following commands should you use to correctly pull the image from the registry?

A. docker pull myregistry.local:5000/myapp

B. docker pull myregistry.local:5000/myapp:latest

C. docker pull myregistry.local:5000/myapp:3.1.4

D. docker image pull myregistry.local:5000/myapp

Prune Images and Containers

Pruning: Overview

Docker provides pruning functionality to help manage unused images efficiently and improve system performance. Pruning involves removing images that are no longer in use.

Pruning can be automated through scheduled tasks or scripts to regularly clean up unused resources, ensuring optimal resource utilization and preventing disk space issues.

Pruning Images

The docker image prune command allows you to clean up unused images. By default, this command cleans the image that is not tagged or referenced by any container.

Command to clean up the unused images

```
$ docker image prune
```

Pruning Containers

Pruning containers involves removing stopped or unused containers from the system. It helps prevent resource wastage and improves system performance by freeing up disk space.

Command to clean up the unused containers or the system

```
 //To clean containers
$ docker container prune

//To clean system by removing unused containers and images
$ docker system prune
```

Best Practices for Docker Pruning

Docker users can follow these best practices to effectively manage resources and maintain system performance over time:

1. Schedule regular pruning
2. Use prune commands
3. Automate pruning tasks
4. Backup consideration
5. Monitoring and optimization

Assisted Practice

Inspecting, Removing, and Pruning Images **Duration: 10 Min.**

Problem statement:

You have been assigned a task to inspect, remove, and prune Docker images to manage Docker image storage efficiently.

Outcome:

By completing this demo, you will be able to inspect Docker images, remove specific images, and prune untagged images, thereby maintaining efficient Docker image management.

Note: Refer to the demo document for detailed steps
08_Inspecting_Removing_and_Pruning_Images

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Inspect, remove, and prune docker images

Real-Life Impacts of Docker Key Components

Docker images serve as the blueprint for creating Docker containers. In actual projects, using standardized and optimized Docker images ensures consistency across different environments (development, testing, production), reducing the *it works on my machine* problem.

Dockerfiles automate the creation of Docker images, allowing teams to define the exact environment needed for their applications. This automation ensures reproducibility and reduces human error.

Docker containers are lightweight, portable units that can run applications consistently across different environments. They have transformed the way businesses deploy applications by allowing faster time-to-market and reducing infrastructure costs.

Real-Life Impacts of Docker Key Components

Tagging allow version control for Docker images, enabling teams to deploy only tested and stable versions of their applications to production. This practice is crucial for reducing the risk of deploying faulty software.

Docker registries (like Docker Hub or private registries) are centralized locations for storing and distributing Docker images. They enable collaboration among teams and ensure that the correct images are used in various stages of the deployment pipeline.

Pruning unused **Docker images** and stopped **containers** frees up disk space and keeps the Docker environment efficient. Companies use automated pruning to manage storage and optimize infrastructure, maintaining system performance.

Quick Check

You have completed a project using Docker and want to clean up your system by removing all stopped containers, unused Docker images, and unused networks to free up space and resources. Which Docker command would you use to accomplish this task efficiently?

A. docker prune

B. docker system prune

C. docker clean

D. docker garbage collect

Key Takeaways

- A Docker image is a file that runs programs within a Docker container. It functions as a set of guidelines like a template.
- A container is the same as a typical operating system process, except that it is isolated and has its own file system, networking, and isolated process tree distinct from the host.
- A container is a standardized software component that wraps up code and its dependencies to ensure that an application runs consistently in different computing environments.
- A Docker registry is a system for storing and distributing Docker images with specific names.
- Docker provides pruning functionality to help manage unused images efficiently and improve system performance.

Dockerizing a Java Program and Setting up a Secure Local Registry

Duration: 25 min.

Project Agenda: To establish a secure local registry and dockerize a Java program for efficient and consistent deployment across various environments

Description: Your company is experiencing a need to streamline deployment processes and ensure consistency across software environments. To address this, you are undertaking a project to dockerize a Java program and set up a local registry with security measures that enhance scalability and maintain deployment consistency.

LESSON-END PROJECT

Dockerizing a Java Program and Setting up a Secure Local Registry

Duration: 25 min.

Perform the following:

1. Write a Java program and create a Dockerfile
2. Create a local Docker registry and run it with an htpasswd file
3. Build and tag the Docker image
4. Log in to the secured registry and push the Docker image to it

Expected deliverables: Dockerized Java program and a secure local registry

LESSON-END PROJECT

Thank You