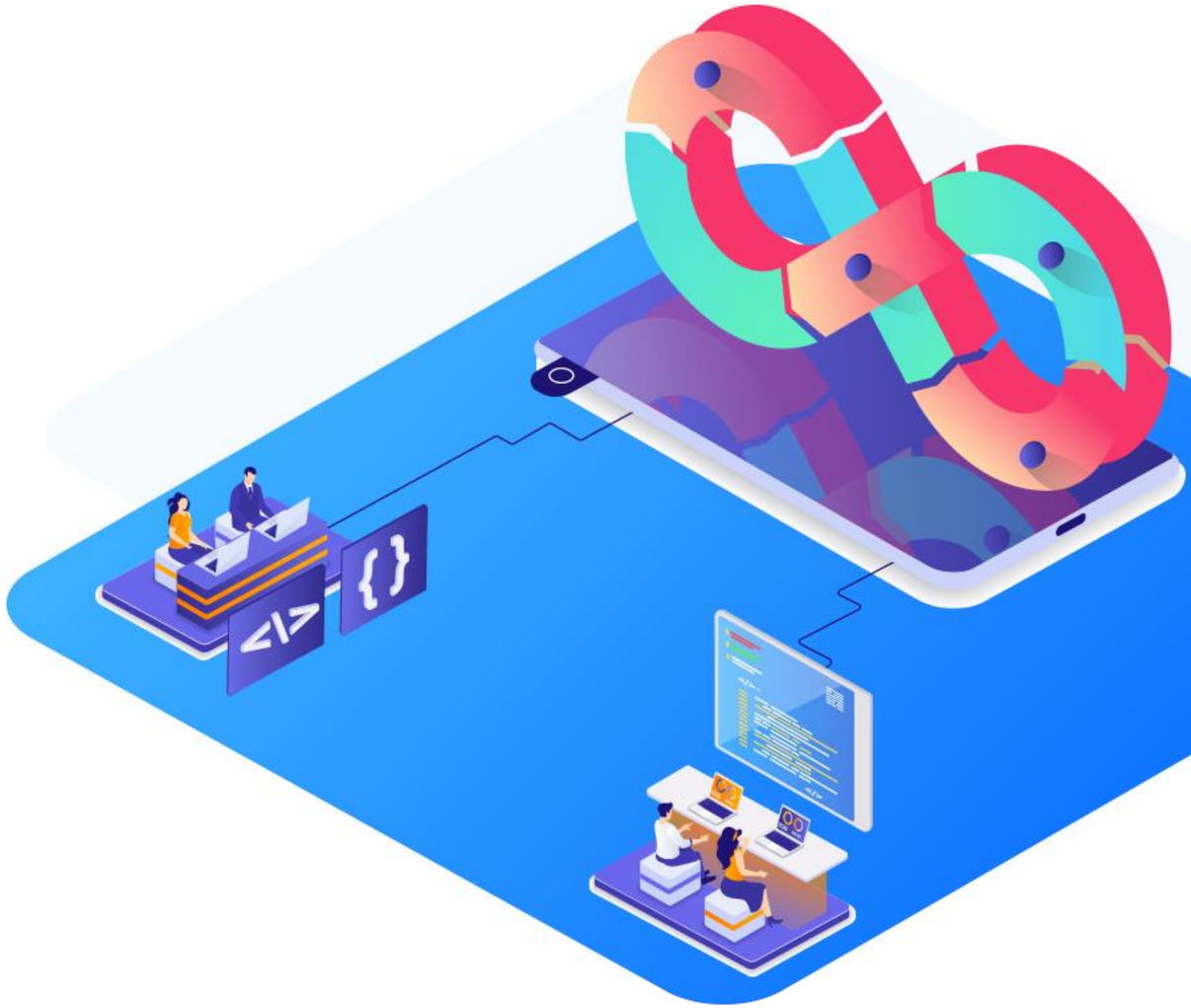
Containerization with Docker



Configuring Docker Storage and Volumes

Learning Objectives

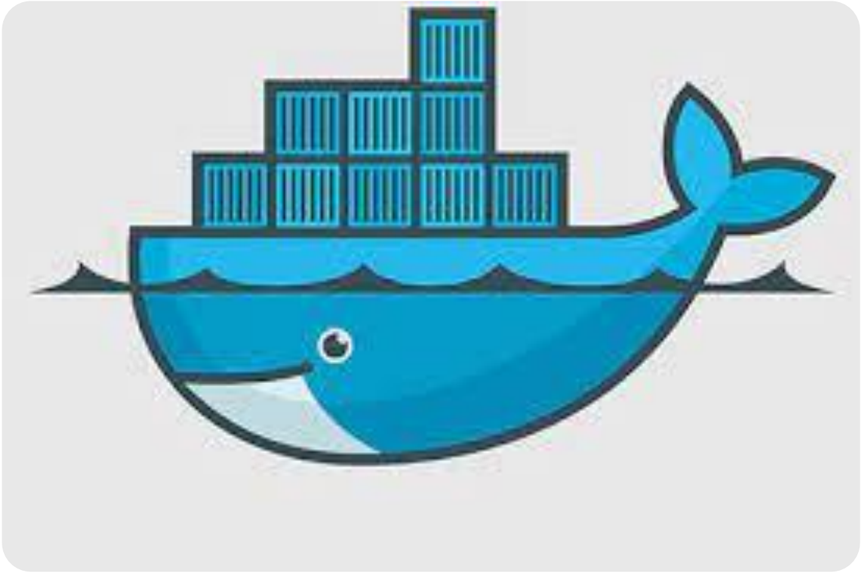
By the end of this lesson, you will be able to:

- Outline the core components of Docker storage for efficient resource management
- Set up persistent storage across multiple nodes to ensure data integrity and performance optimization
- Create and manage Docker volumes effectively to optimize data management and performance.
- Utilize DeviceMapper for efficiently troubleshoot issues within Docker environments

Introduction to Storage in Docker

Docker Storage: Overview

Docker storage refers to the mechanisms Docker provides for managing data within containers and persisting it beyond container lifecycles.



Common storage solutions include volumes, bind mounts, and tmpfs mounts, each designed to meet different needs for data persistence and sharing between containers and hosts.

Docker Storage: Benefits

Docker storage offers several benefits, including:

Consistent environment

Docker ensures consistency across environments, reducing compatibility issues and simplifying deployment.

Faster deployment

Docker enables fast deployment and scaling of applications, enhancing agility in development and operations.

Efficient management

Docker facilitates efficient management of multi-cloud environments, streamlining operations and enhancing flexibility.

Docker Storage: Benefits

Enhanced security

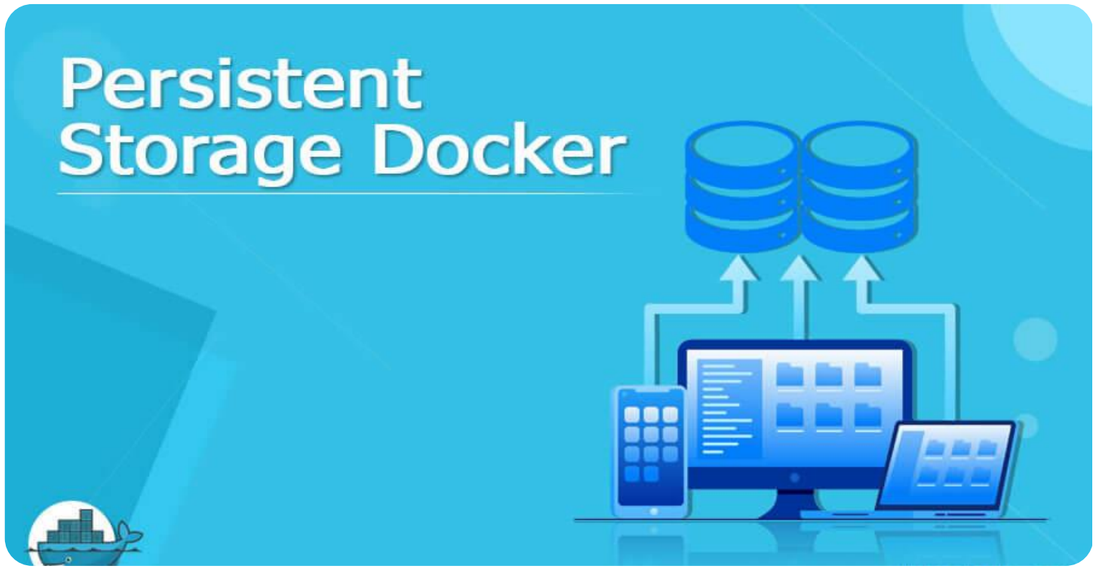
Docker containers offer enhanced security through isolation and resource control, reducing the risk of system vulnerabilities.

Optimized costs

Docker helps optimize costs by reducing resource overhead and enabling efficient resource utilization.

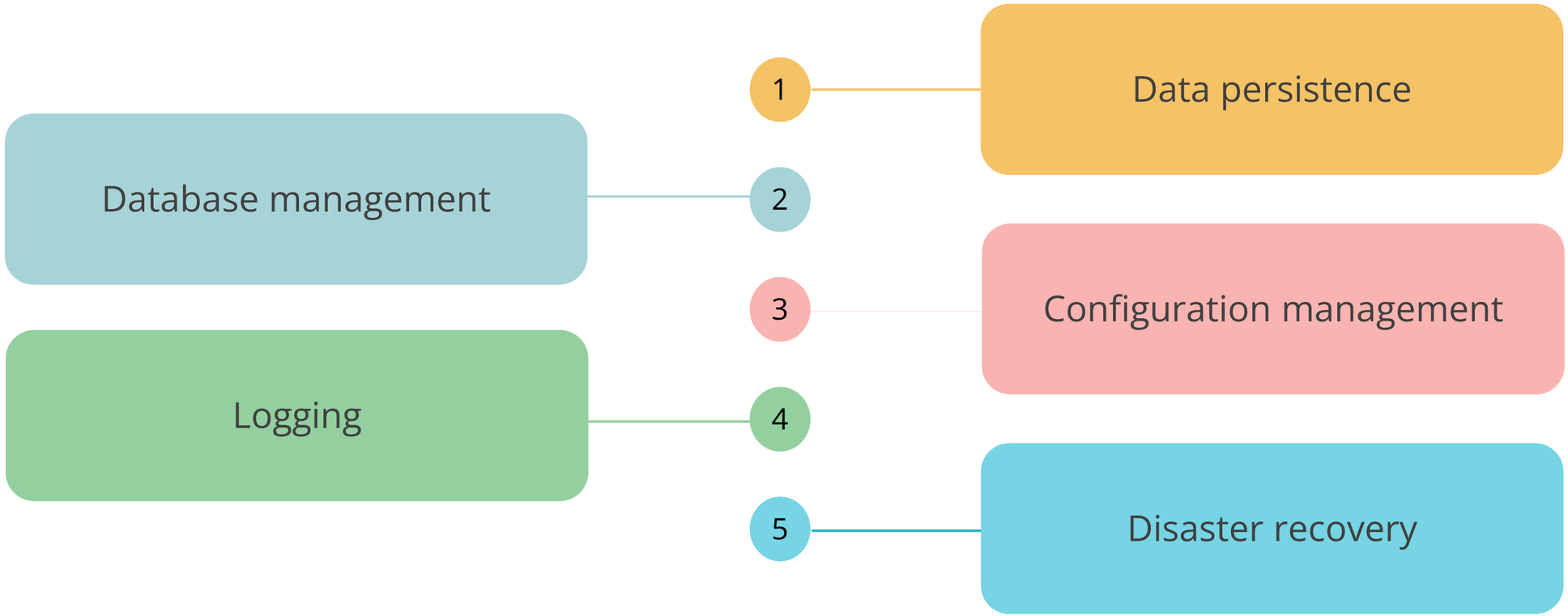
Persistent Storage

It refers to the ability to retain data beyond the lifecycle of containers which allows data to persist even if the container is stopped, removed, or replaced.



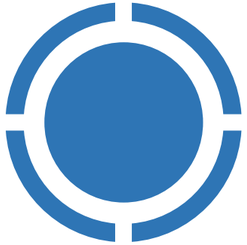
This is vital for apps needing data accessibility across containers or safeguarding critical information from loss.

Persistent Storage: Use Cases

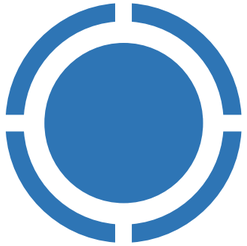


Persistent Storage: Methods

Below are the two main methods to achieve persistent storage:



Volumes: Docker volumes are the preferred mechanism for persistent storage in Docker.



Bind mounts: Bind mounts are a storage option in Docker that involves mapping a specific file or directory on the host machine to a path in the container.

Assisted Practice

Demonstrating how to use storage across cluster nodes **Duration: 10 Min.**

Problem statement:

You have been assigned a task to demonstrate the use of storage across cluster nodes by utilizing a volume, replicating a service, and inspecting worker nodes in Docker.

Outcome:

By completing this demo, you will be able to utilize a volume for storage, replicate a service across multiple nodes, and inspect worker nodes to ensure proper data persistence and container management across a Docker cluster.

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

Assisted Practice: Guidelines

Steps to be followed:

1. Utilize a volume for storage, replicate a service, and inspect worker nodes

Quick Check

As a DevOps engineer, you are tasked with setting up an automated pipeline for a software project that involves building, running, and distributing Docker containers. Which Docker component would you primarily interact with to manage data effectively?

A. Docker daemon

B. Images

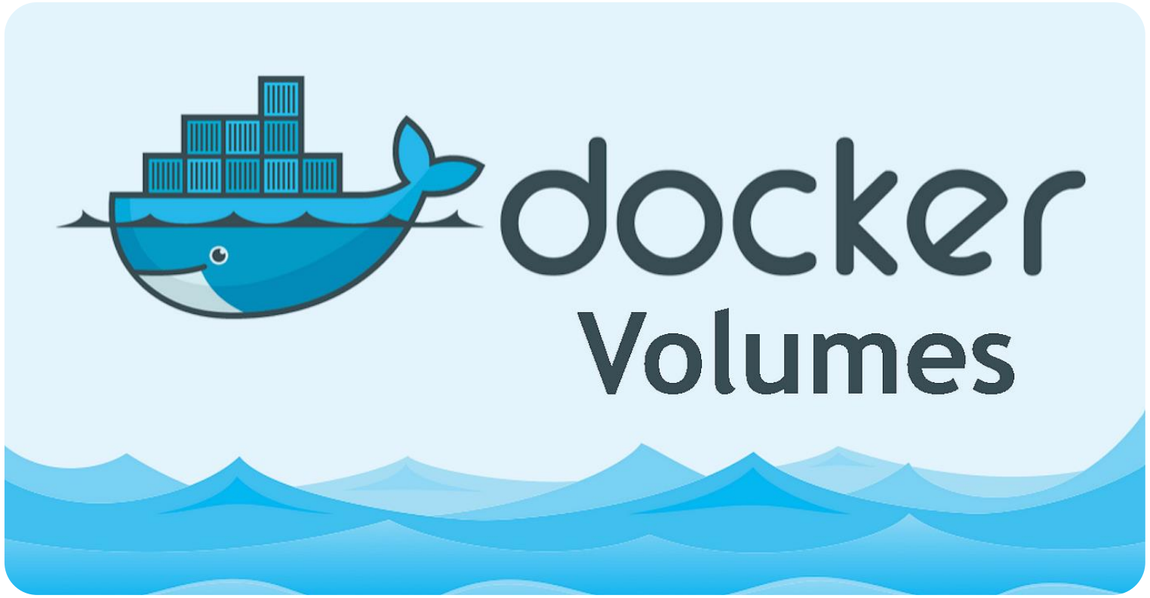
C. Docker host

D. Storage

Docker Volumes and Bind Mounts

Docker Volumes: Overview

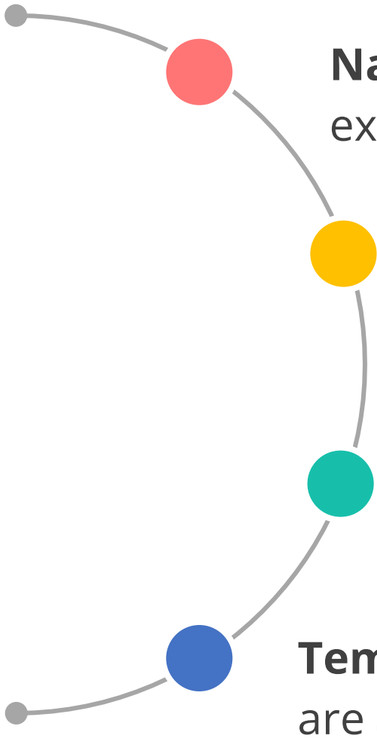
Docker volumes are a fundamental feature used for persistently storing and managing data in containers.



Volumes are created and managed by Docker, simplifying the process of data storage management within containers.

Types of Volumes

Here are the different types of volumes:



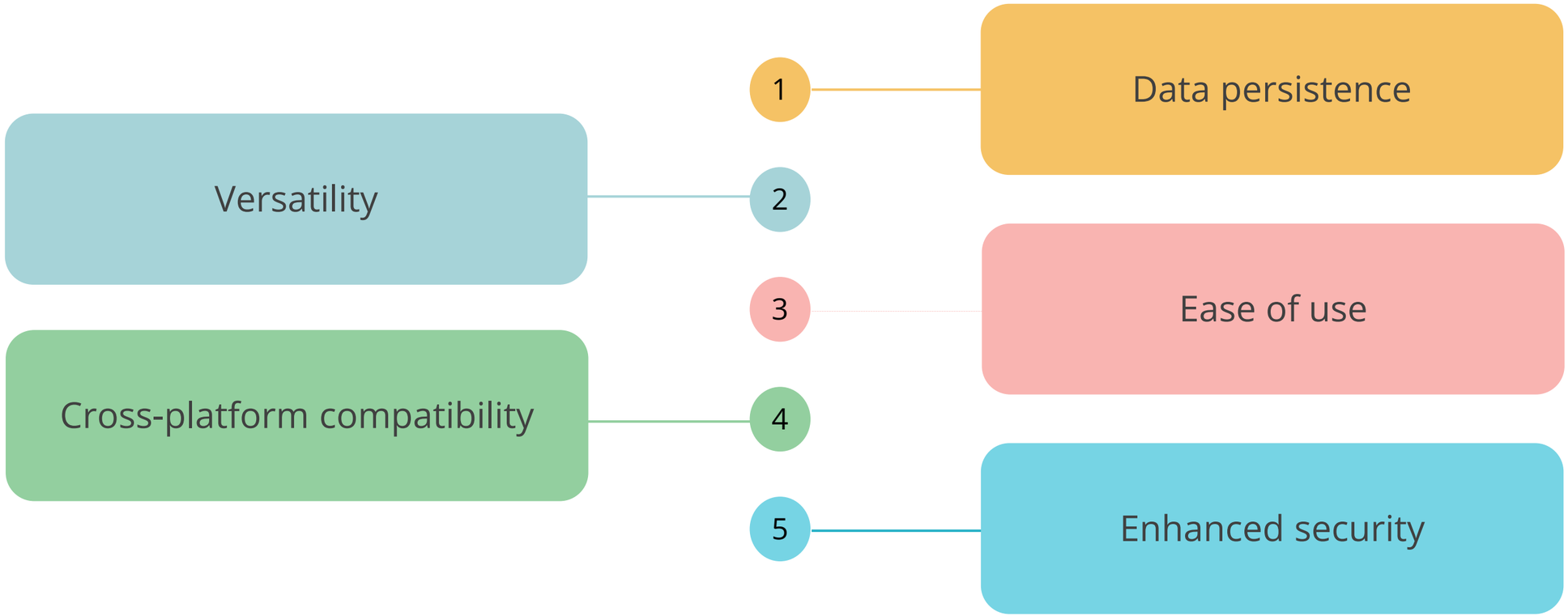
Named volumes: These are managed by Docker and can be given explicit names.

Anonymous volumes: These are useful when the container doesn't need persistent storage.

Host volumes: These provide a way to persist data from the container to the host filesystem, making it accessible outside of the container's environment.

Temporary volumes (tmpfs): These are not stored on the host filesystem and are automatically removed when the container stops.

Volumes: Benefits



Creating Docker Volumes

Users can run the following command to create a Docker volume:

```
docker volume create [volume_name]
```

Note: This command should be run in the terminal or command-line interface (CLI) of the host machine where Docker is installed. Before executing the command, ensure Docker is properly set up and running.

Bind Mounts: Overview

Bind mounts are a type of storage that allows the sharing of files or directories between the host system and Docker containers.

Developers often use bind mounts during development to share code or data between their local environment and Docker containers.

They can be particularly useful for accessing configuration files, sharing application code, or persisting data.

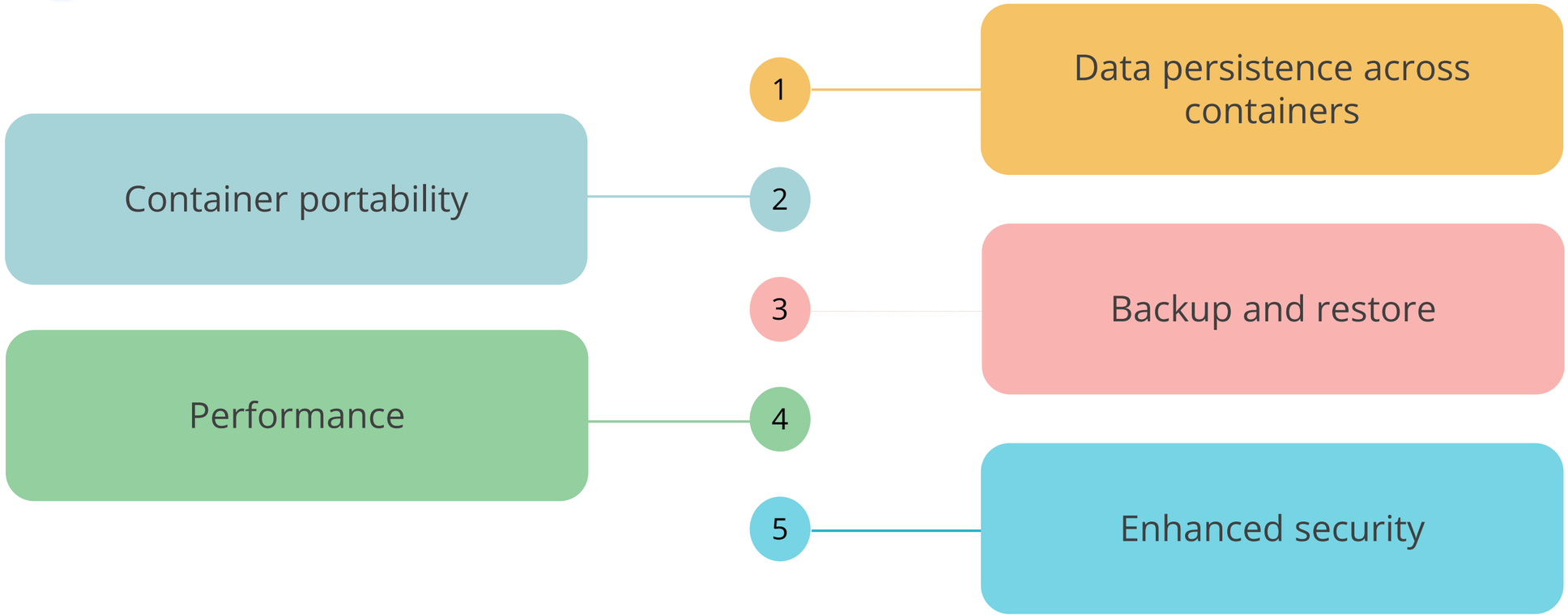
Mount Volumes to Containers

Mounting volumes to Docker containers is crucial for persisting data and enabling seamless data exchange between the host and containers.

```
docker run -v <host_path>:<container_path> <image_name>
```

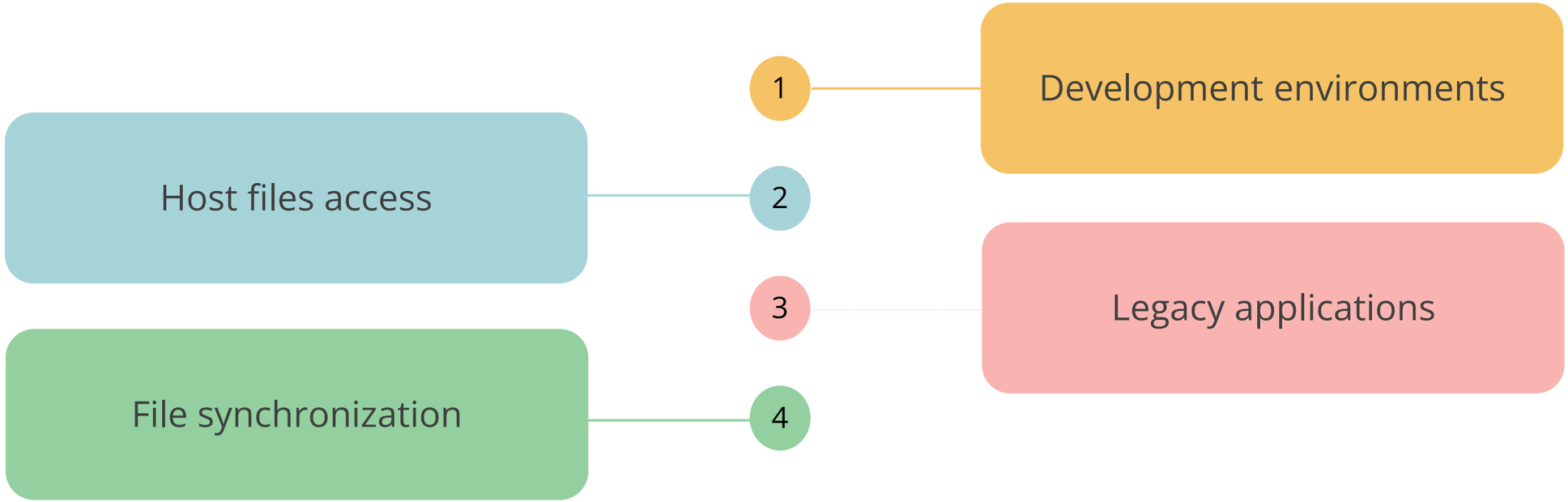
To mount a Docker volume, you can use either the **-v** or **--volume** flag with the docker run command.

When to Use Docker Volumes?



- Use Docker Volumes to persist data across containers, ensure container portability, or want better security and performance

When to Use Bind Mounts?



- Use Bind Mounts when you need direct access to host files, are in a development environment, or need real-time synchronization between the host and container

Best Practices for Using Docker Volumes

Users can effectively manage Docker volumes and optimize resource utilization by adhering to the following best practices:

1. Limit the number of volumes and use Named volumes
2. Cleanup unused volumes regularly
3. Backup data frequently
4. Manage volume permissions
5. Monitor volume usage

Assisted Practice

Creating a Docker volume and mounting it to a container **Duration: 10 Min.**

Problem statement:

You have been assigned a task to create a Docker volume and mount it to a container for better data management and containerized application deployment.

Outcome:

By completing this demo, you will be able to create a Docker volume, mount it to a container, and verify the volume attachment, enhancing data management capabilities.

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

Assisted Practice: Guidelines

Steps to be followed:

1. Create and mount a Docker volume to a container

ASSISTED PRACTICE

Quick Check

You are setting up a containerized database for your application, and you need to ensure that the database data persists even if the container is removed or updated. Which storage option should you choose?

A. Bind Mount

B. Docker Volume

C. tmpfs Mount

D. Container Layer

Backup and Restore Strategies

Importance of Data Backup in Docker

Disaster recovery

Backups enable quick restoration of Dockerized applications and their data in case of loss or corruption.

Data integrity

Regular backups aid in data integrity by enabling recovery from accidental deletions, errors, or system failures.

Versioning

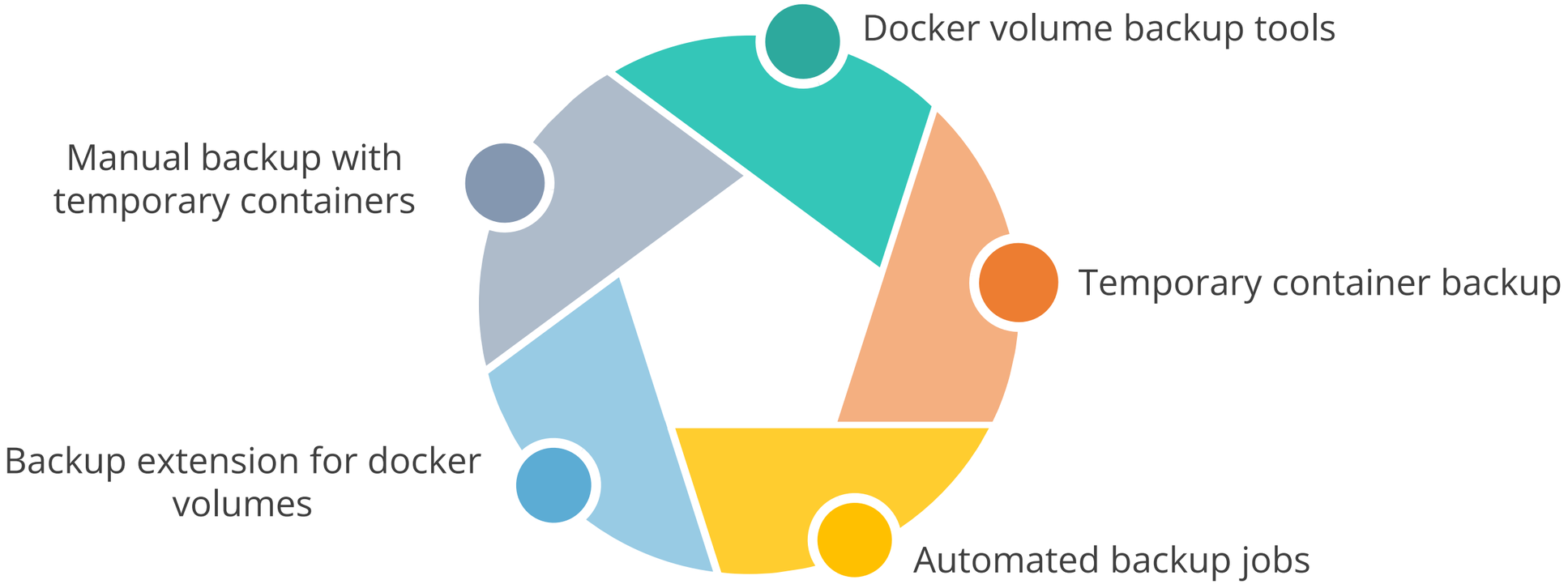
Tracking version numbers in backups enables rollback to prior states as a safety net during updates or changes.

Compliance requirements

Industries must adhere to regulations requiring data backup and retention which can ensure compliance.

Backup for Docker Volumes

Users can effectively safeguard the Docker volumes by employing the following backup strategies or tools:



Restoring Data from Backups

Users can follow these steps to restore data from backups in Docker:

Identify backup location

Determine the storage location for Docker volume backups, which may include a local directory, remote server, or cloud storage.

Access backup files

Access backup files for restoration, which may require volume mounting or accessing a repository based on your backup approach.

Stop containers

Stop running containers associated with the data to avoid conflicts during restoration.

Restoring Data from Backups

Users can follow these steps to restore data from backups in Docker:

Restore backup

Use Docker commands or backup restoration tools to restore the data from the backup files to the appropriate Docker volumes.

Start containers

Once the data is restored, start the Docker containers that rely on the restored data volumes.

Verify restoration

Verify that the data has been successfully restored by checking the functionality of the applications or services running in the Docker containers.

Assisted Practice

Developing a Backup and Restore Plan for Docker Volumes **Duration: 10 Min.**

Problem statement:

You have been assigned a task to create, backup, and restore Docker volumes to ensure data resilience and facilitate efficient disaster recovery strategies.

Outcome:

By completing this demo, you will be able to create a Docker volume, back it up, and restore it to a new container, ensuring data resilience and efficient disaster recovery.

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

Assisted Practice: Guidelines

Steps to be followed:

1. Create, backup, and restore a Docker volume

Quick Check

Your team needs to back up the data from a running container’s storage and ensure it can be restored later in case of failure. Which Docker command should you use to perform this task?

A. docker cp

B. docker commit

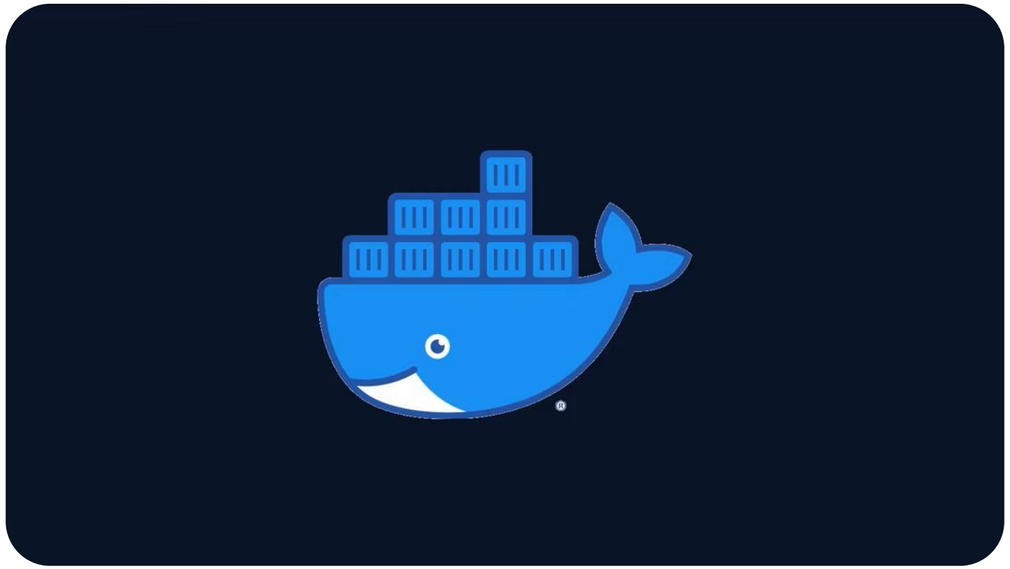
C. docker volume create

D. docker run --mount

Storage Drivers in Docker

Storage Drivers: Overview

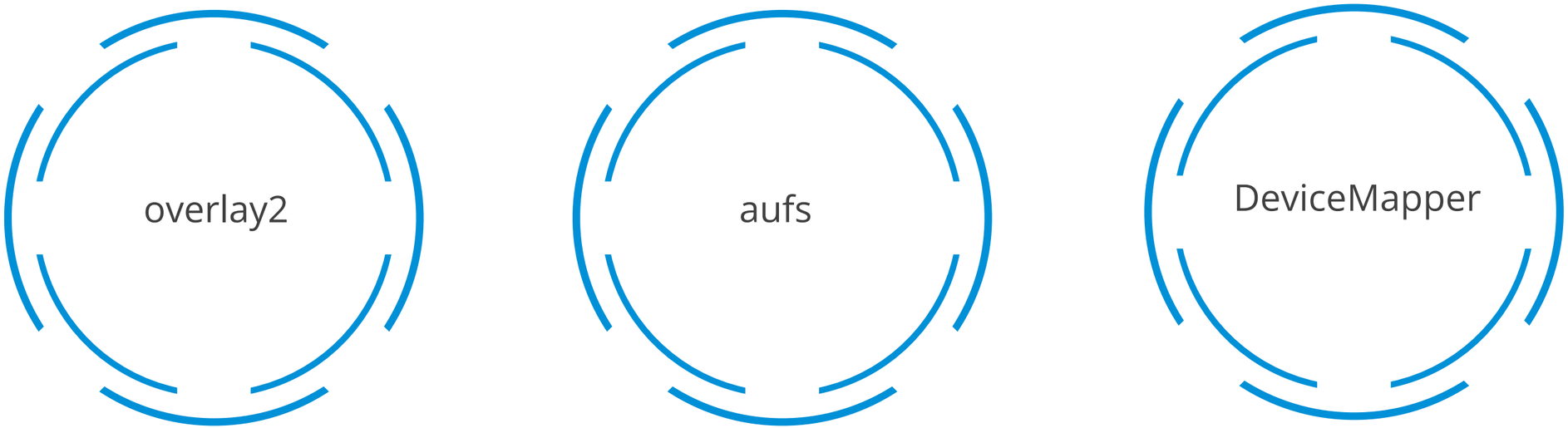
Storage drivers in Docker are essential components that manage how images and containers are stored and handled on a Docker host.



These drivers manage data storage and retrieval for container execution.

Storage Drivers: Types

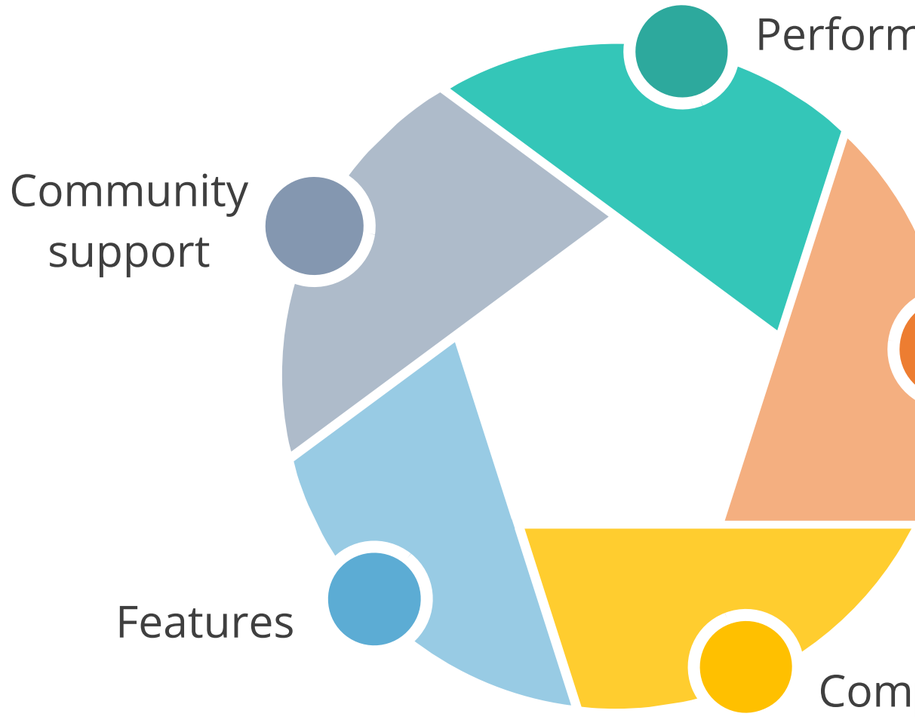
Docker provides various storage drivers to manage image and container storage on a Docker host, which are:



Choosing the Right Storage Driver

Choosing the right Docker storage driver is essential for peak performance and workload compatibility.

The essential factors in selecting a storage driver include:



Storage Driver: Performance Considerations

When choosing storage drivers in Docker, performance considerations are crucial for optimal container operation.

Speed efficiency vs. write speed

Storage drivers optimize space efficiency, but write speeds may vary, especially compared to native file systems.

Performance assessment

Evaluate each storage driver's performance in your environment based on workload needs, as different drivers excel in different scenarios.

Compatibility and infrastructure

Select storage drivers that balance compatibility with your infrastructure and meet performance needs without sacrificing compatibility.

Quick Check

You are deploying Docker on a system with a high-performance SSD and need to optimize container storage for speed and efficiency. Which Docker storage driver would be most suitable for this setup?

A. overlay2

B. aufs

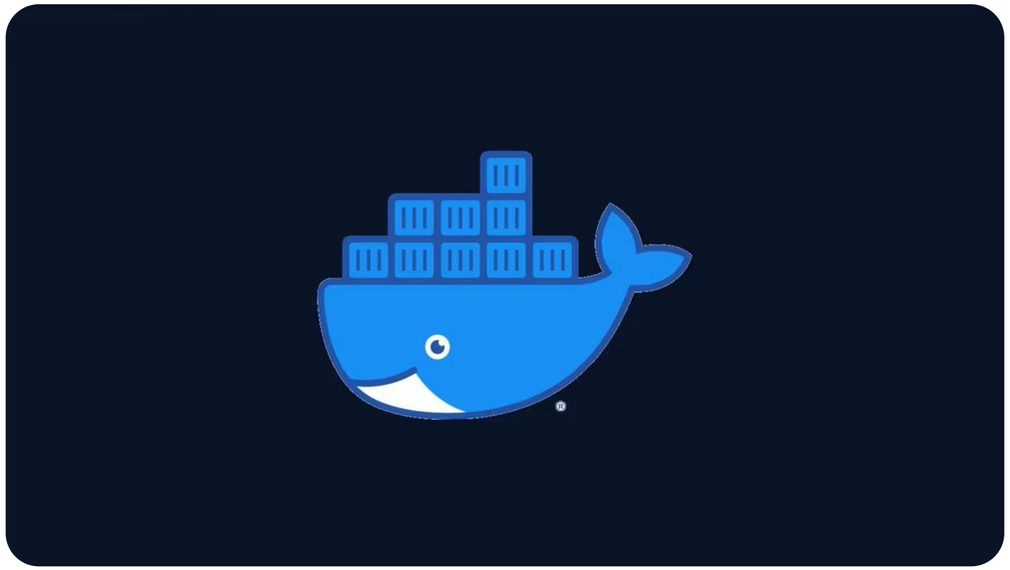
C. DeviceMapper

D. vfs

DeviceMapper in Docker

DeviceMapper: Overview

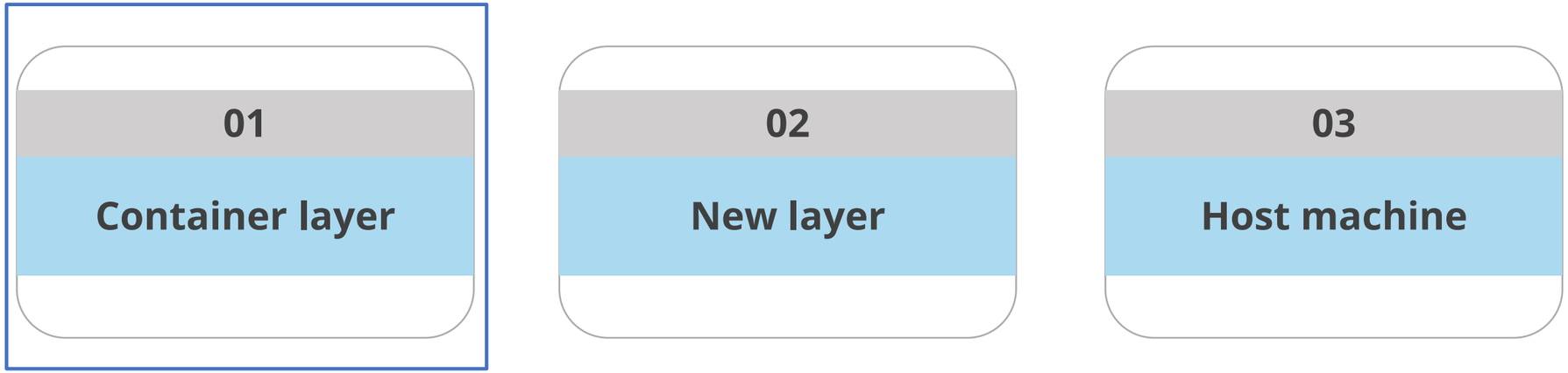
The DeviceMapper storage driver is utilized in Docker for image and container management.



It relies on the Device Mapper framework, which is kernel-based and supports various advanced volume management technologies on Linux.

DeviceMapper: Workflow

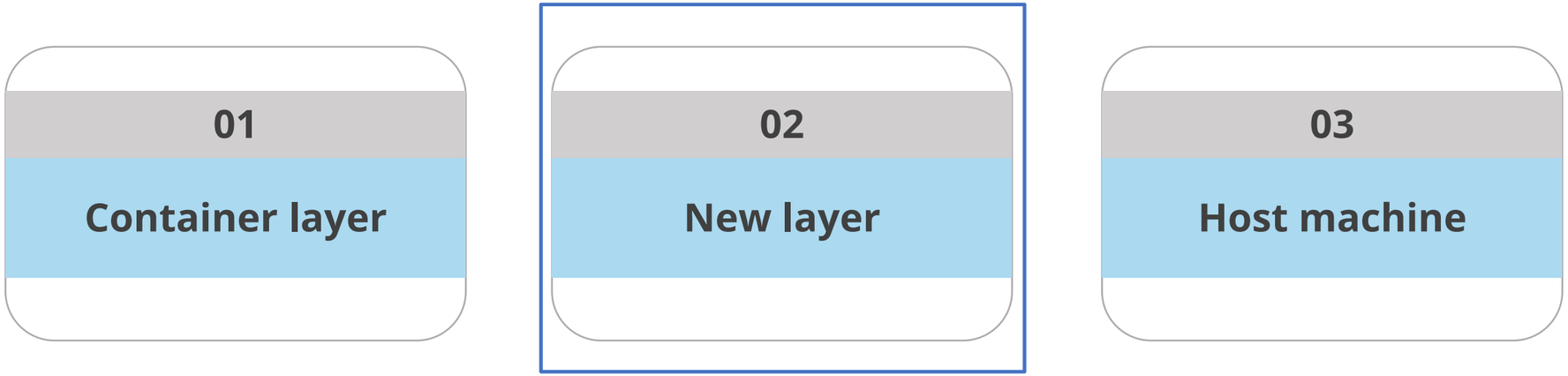
In the DeviceMapper workflow for Docker, the **container layer** refers to the top writable layer of a Docker container.



This allows the container to make changes or write new data without altering the underlying image layers, ensuring that each container can have its own unique data set and state independent of its base image.

DeviceMapper: Workflow

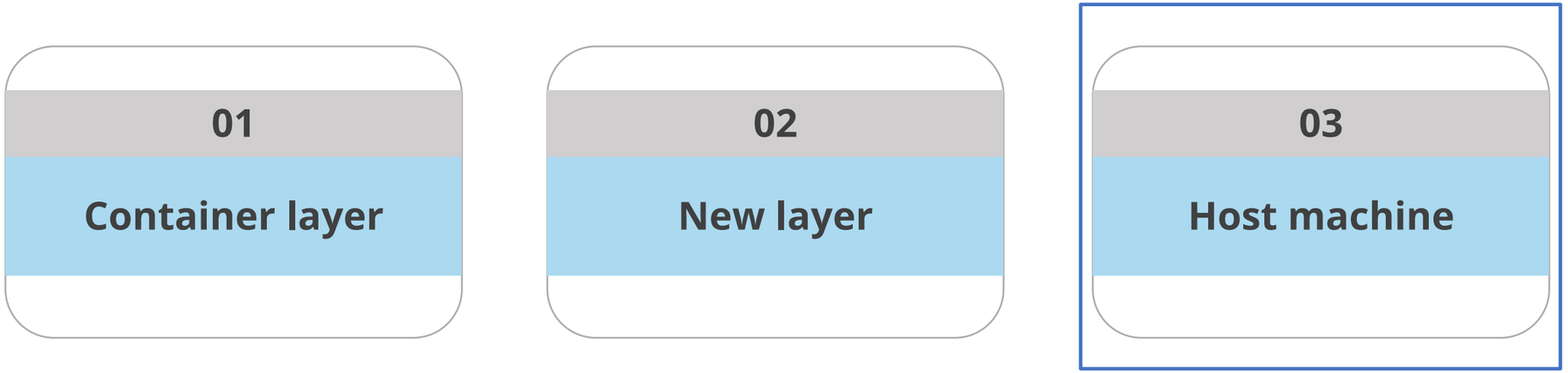
In the DeviceMapper workflow for Docker, the **New layer** refers to the additional layer created when a container is modified.



When a new layer is created (like when a container is modified), DeviceMapper creates a copy-on-write snapshot of the previous layer.

DeviceMapper: Workflow

In the DeviceMapper workflow for Docker, the **host machine** refers to the physical or virtual machine that runs the Docker engine and hosts the Docker containers.



All these devices are stored in the **/var/lib/docker/devicemapper/** directory on the host machine.

DeviceMapper and Docker Performance

DeviceMapper storage driver can have a slight performance impact on Docker.

Allocate-on-demand

The DeviceMapper storage driver utilizes this operation to allocate new blocks from the thin pool to a container's writable layer.

Copy-on-write

This operation is used when a container modifies a specific block for the first time. Then, that block is written to the container’s writable layer.

Assisted Practice

Selecting a Storage Driver **Duration: 10 Min.**

Problem statement:

You have been assigned a task to configure different Docker storage drivers, specifically overlay2 and fuse-overlayfs, to optimize storage management and enhance system performance.

Outcome:

By completing this demo, you will be able to configure Docker to use the overlay2 and fuse-overlayfs storage drivers, ensuring optimized storage management and improved system performance.

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

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

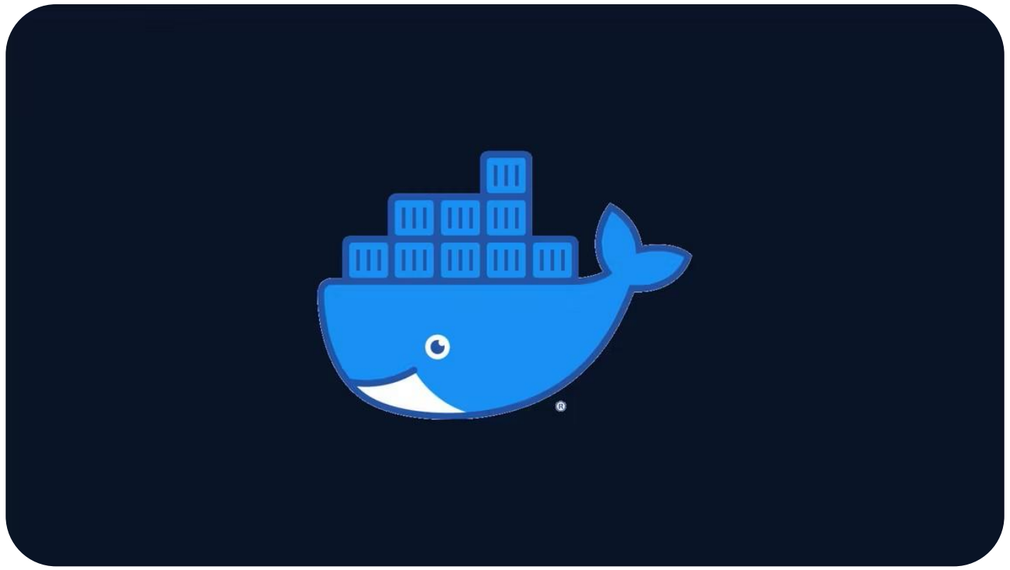
1. Configure overlay2 storage driver
2. Configure fuse-overlayfs storage driver

ASSISTED PRACTICE

Graph Drivers in Docker

Graph Drivers: Overview

Graph drivers in Docker are essential components responsible for managing the storage and retrieval of Docker images and container filesystems.



These drivers play a crucial role in how images and containers are stored and managed on the Docker host.

Graph Drivers: Features

Pluggable architecture

Docker supports multiple graph drivers using a pluggable architecture, allowing users to select the appropriate driver based on their requirements and environment.

Efficient image layering

Graph drivers efficiently manage image layering, allowing Docker to pull, build, and store images in a manner that minimizes storage overhead and optimizes performance.

Customization

Users can customize graph drivers to suit specific storage needs or integrate with external storage solutions, enhancing flexibility and adaptability.

Identify the Correct Graph Driver

The following are the ways to identify the correct graph drivers for different operating systems for ensuring optimal performance and compatibility:

01 Ensure that the graph drivers you choose are compatible with your system’s hardware configuration.

02 Check the application's documentation or support resources for any guidance on selecting compatible drivers.

03 Keep your graph drivers up to date to benefit from performance improvements, bug fixes, and security patches.

Key Takeaways

- Docker Engine is a fundamental component of the Docker platform, providing the core containerization technology for building, containerizing, and running applications.
- Docker is built on a client-server model, which includes the three main components that are Docker Client, Host, and Registry.
- 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 improving system performance.

Containerizing a Legacy Application

Duration: 25 min.

Project Agenda: To containerize a legacy Python application using Docker, integrating diverse storage and volume strategies for ensuring data persistence and optimizing performance

Description: Your company is experiencing the challenge of modernizing legacy applications while ensuring portability and streamlined deployment processes. To address this, you are undertaking a project that aims to containerize legacy applications using Docker, integrating a variety of storage and volume strategies.

LESSON-END PROJECT

Containerizing a Legacy Application

Duration: 25 min.

Perform the following:

1. Create a Dockerfile and define the Python dependencies
2. Create the Django project using Docker Compose
3. Set up database connection and configure Docker Compose
4. Change the ownership of files and start the application
5. Verify the setup and clean up the environment

Expected deliverables: Containerized legacy application integrated with storage and volume strategies

LESSON-END PROJECT

Thank You