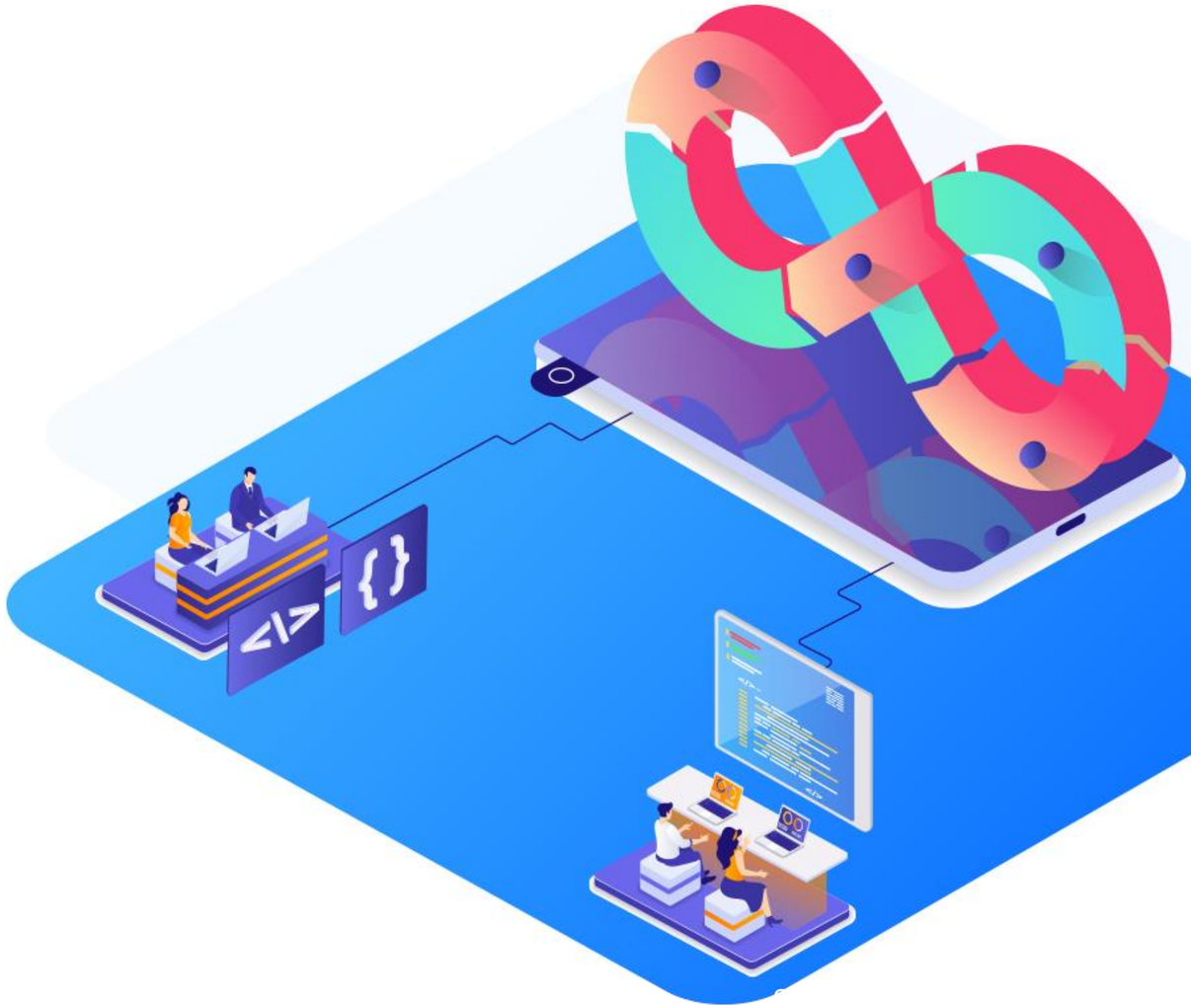
Containerization with Docker



Docker Introduction

Learning Objectives

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

- Identify the features of Docker to deploy applications efficiently
- Install Docker to streamline the development workflow by providing a consistent environment for deploying applications
- Utilize and configure the Mirantis products to set up Docker for enterprise purposes

Docker: Overview

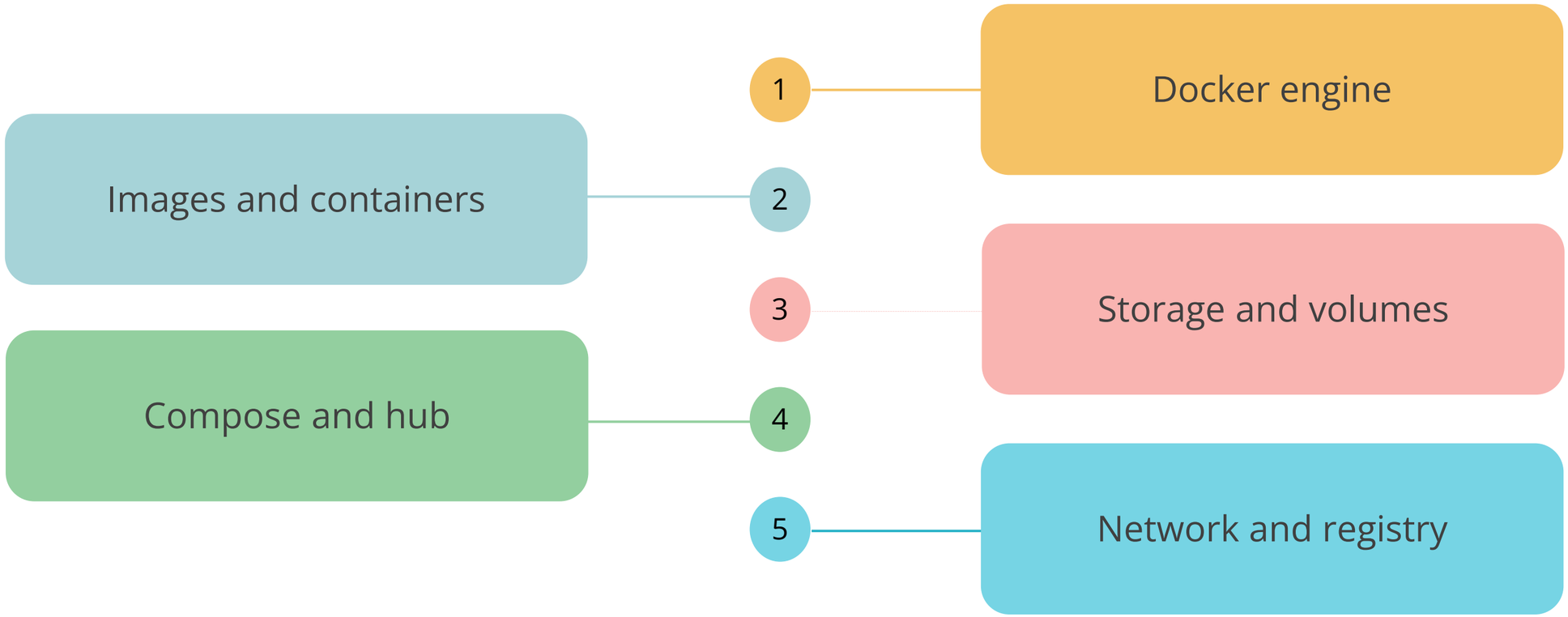
What Is Docker?

Docker is an open-source containerization platform designed to simplify the development, deployment, and management of applications.

Docker helps manage application infrastructure using various concepts, such as images, containers, Swarms, and microservices.

Docker: Components

Docker offers several vital components, as shown below, to optimize application development, deployment, and management. Containerization maintains consistent environments across all stages, enabling rapid, efficient, and portable software delivery.



Docker: Components

Docker engine	Docker Engine is the core software that enables containerization. It allows you to build, run, and manage containers on a host system.
Images and containers	Docker images are read-only templates containing the application code and environment, while containers are runtime instances of these images that execute in isolated environments.

Docker: Components

Storage and volumes	Docker storage manages data persistence for containers. Volumes store data outside the container's filesystem, ensuring it persists across restarts and re-creations.
Compose and hub	Docker Compose is a tool for defining and managing multi-container Docker applications. Docker Hub is a cloud-based registry service for storing, sharing, and distributing Docker images.
Network and registry	Docker networks enable secure, isolated communication between containers. Docker registries, like Docker Hub or private registries, store and distribute Docker images.

Why Use Docker?

Here are some of the reasons to use Docker:

Consistency Across Environments

Docker ensures that your application works seamlessly in any environment from development through staging to production.

Rapid Deployment

Containers can be spun up in seconds, much faster than deploying applications on physical or virtual machines.

Microservices Architecture

Docker suits microservices architectures by allowing each component to run in a separate container with all dependencies.

Scalability

Containers can be easily started, stopped, or replicated across hosts to scale out or handle increased load.

Business Use Cases of Docker

01 Healthcare

Healthcare organizations utilize Docker to securely handle sensitive patient data and support reproducible research environments that maintain data privacy.

02 IT industries

IT firms use Docker for consistent development and testing environments, promoting rapid and scalable software deployment.

Business Use Cases of Docker

03 Finance

Financial institutions leverage Docker for enhanced security and compliance, and to support microservices architectures that improve system modularity and downtime reduction.

04 Retail

Retail companies deploy Docker to manage seasonal traffic spikes efficiently and to run sophisticated machine learning models for customer personalization.

Business Use Cases of Docker

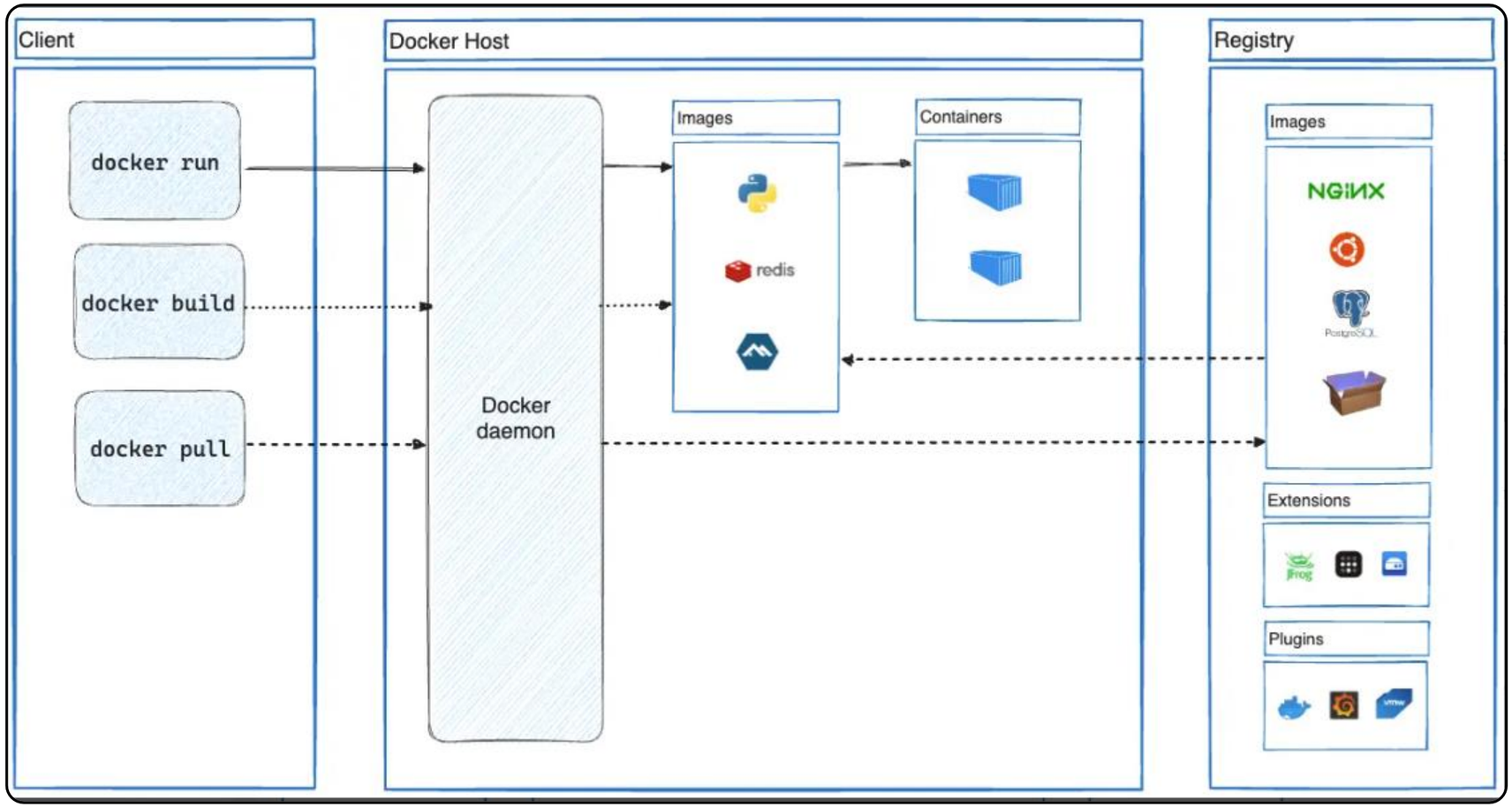
05 Telecommunications

Telecom companies use Docker for network function virtualization to reduce hardware costs and enhance service deployment and scalability.

Docker Architecture and Installation

Docker Architecture

Docker is built on a client-server model consisting of three main components:
Docker Client, Host, and Registry.



Docker Architecture

Docker Client

- The Docker client allows users to interact with Docker through commands like docker run, docker build, and docker pull. These commands are sent to the Docker daemon for execution.

Docker Host

- Consists of Docker daemon that manages Docker objects like images and containers.
- Images are read-only templates containing all dependencies
- Containers are run-time instances of the Docker image.

Docker Registry

- Images in the registry are stored in repositories in the registry. **docker pull** command is used to fetch the images from the registry.
- Extensions and Plugins extend Docker's functionality, allowing integration with various tools and platforms.

Docker Architecture

Here is an explanation of the workflow of the Docker architecture, explaining the interaction of client with host machine:

docker build: The client sends a build command to the Docker daemon, which constructs an image based on the instructions in a Dockerfile and stores it on the Docker host.

docker pull: The client requests an image from the registry. The daemon retrieves the image and stores it locally.

docker run: The client instructs the daemon to run a container from an image. The daemon creates and starts the container, using the specified image.

Components of Docker Architecture

Docker Client

This is the interface through which users interact with Docker. It sends requests to the Docker daemon.

Docker Daemon

The daemon is responsible for building, running, and distributing Docker containers.

Docker Host

It is a machine (physical or virtual) running the Docker daemon.

Docker Registry

It holds different versions of images and facilitates image sharing across different environments.

Components of Docker Architecture

Images

These are independent packages that contain all the essentials for running software.

Containers

They function independently, leveraging the host machine's OS kernel, with distinct file systems.

Networks

Docker allows creating isolated networks to connect containers and control their communication.

Storage

Docker offers storage choices, such as volumes and bind mounts, to store data produced by containers.

Docker Engine

It is a fundamental component of the Docker platform that provides the core containerization technology for building, containerizing, and running applications.

It provides platform independence, enabling users to develop and run containers consistently across different environments. Docker engine works like a package manager and operates internally to facilitate the necessary components for Docker Daemon to work on Host

Installing Docker

1. Install packages and update the apt package index to enable apt to access an HTTPS repository. Make sure you have a minimum of Ubuntu 20.04 of 64-bit with at least 2GB RAM:

Example:

```
sudo apt-get update

$ sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

Installing Docker

2. Add Docker’s official GPG key:

Example:

```
sudo mkdir -p /etc/apt/keyrings
$ curl -fsSL
https://download.docker.com/linux/ubuntu/gpg | sudo
gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

Installing Docker

3. Use the following command to set up the repository:

Example:

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Installing Docker

4. Install the package with apt as follows:

Example:

```
sudo apt-get update

sudo apt-get install ./docker-desktop-<version>-
<arch>.deb
```

Installing Docker

5. Launch Docker Desktop for Linux by searching for it in the **Applications** menu and clicking on it

Example:

```
systemctl --user start docker-desktop
```

Assisted Practice

Configure Docker Daemon to start on boot **Duration: 10 Min.**

Problem statement:

You have been assigned a task to configure the Docker Daemon on an Ubuntu system to start on boot and manage Docker settings.

Outcome:

By completing this demo, you will be able to configure the Docker Daemon to start on boot and customize Docker settings to manage disk space for Docker images and containers.

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

Assisted Practice: Guidelines

Steps to be followed:

1. Configure Docker Daemon
2. Start and configure Docker service using the upstart command

ASSISTED PRACTICE

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 consisting of three main components: Docker Client, Host, and Registry.
- The industries that use Docker are Finance, Healthcare, Telecommunications, Software, and IT industries.
- Docker Client is the interface through which users interact with Docker. It sends requests to the Docker daemon.
- Docker offers storage choices, such as volumes and bind mounts, to store data produced by containers.

Thank You