DevOps Certification Training

TECHNOLOGY

Containerization with Docker

simplilearn

Learning Objectives

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

- Install Docker on your system
- Describe Docker image and containers
- Manipulate container with Docker Client
- Build Custom Image through Docker Server
- Implement Docker Compose with Multiple Local Containers

TECHNOLOGY

Introduction to Docker

What Is Docker?

- Docker is a platform for developers and sysadmins to develop, ship, and run applications by using containers.
- Docker helps the user to quickly assemble applications from its components and eliminates the friction during code shipping.
- Docker aids the user to test and deploy the code into production.

Functionalities

Test the application

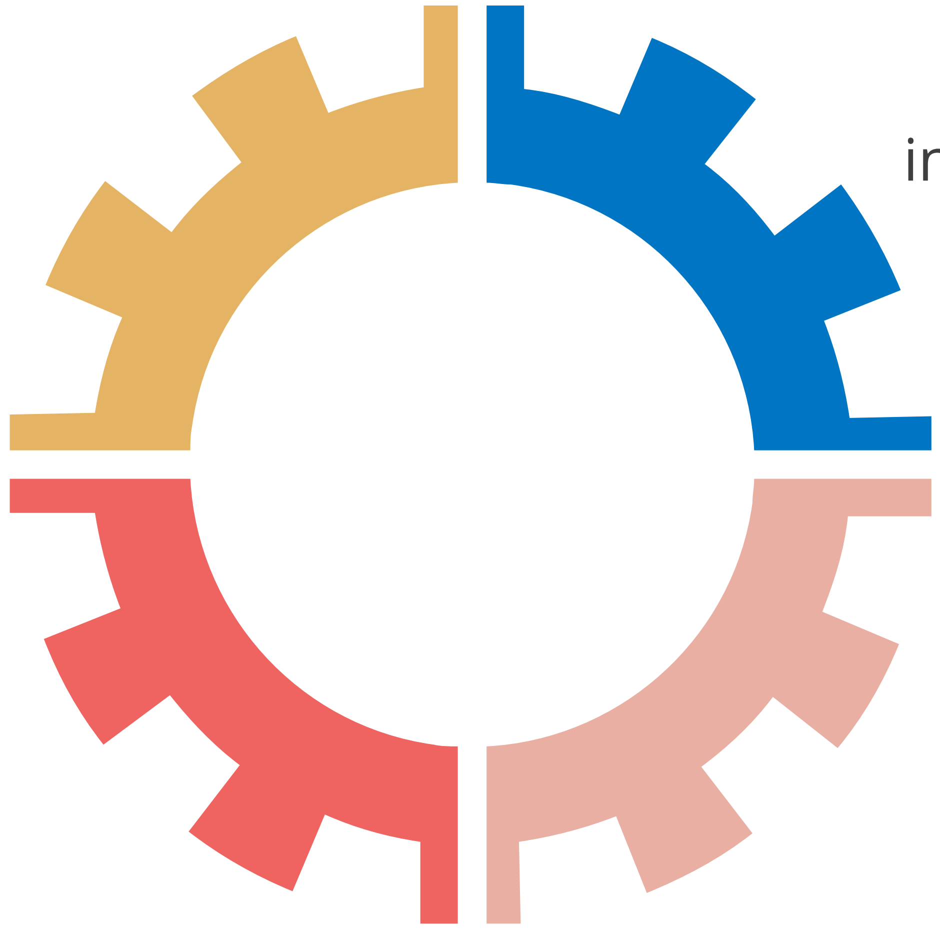
Develop application and its supporting components using containers

Docker provides a platform for the user to:

Deploy the application into production environment, as a container or an orchestrated service

Why Use Docker?

Simplifies and accelerates workflow



Provides freedom to innovate application by using choice of tools

Has high density and runs more workloads

Aids in quick deployment for easy management

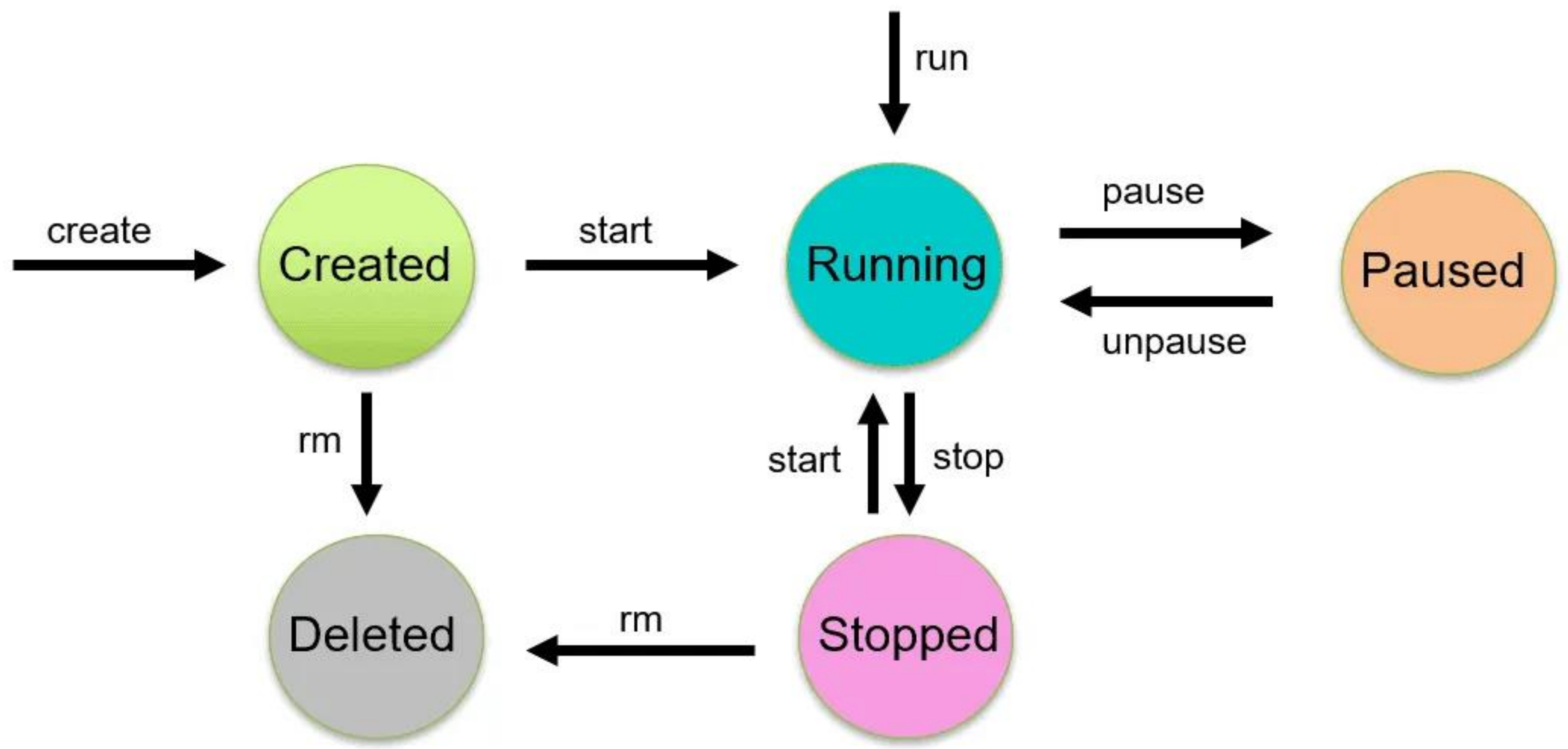
Docker Client

- The **Docker client** is the primary way that many **Docker** users use to interact with **Docker**.
- When you use commands such as **docker** run, the **client** sends these commands to dockerd, which carries them out.
- The **docker** command uses the **Docker** API.
- The **Docker client** can communicate with more than one daemon.

Components of Docker

Docker Lifecycle

Docker lifecycle has various stages as shown below:



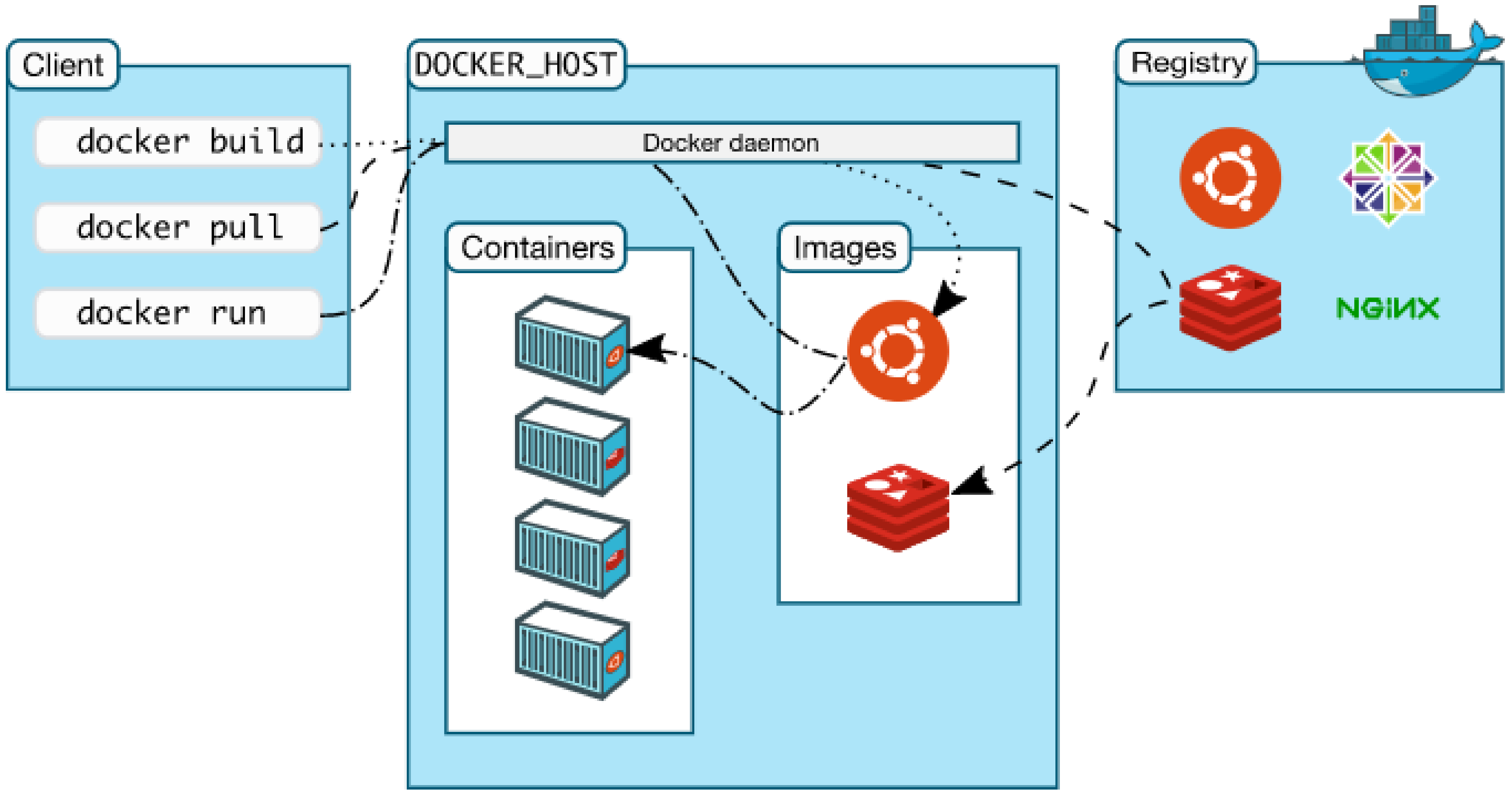
Docker Lifecycle

Let's understand what each stage in the life cycle does.

- **Created**: A container that has been created but not started
- **Running**: A container running with all its processes
- **Paused**: A container whose processes have been paused
- **Stopped**: A container whose processes have been stopped
- **Deleted**: A container in a dead state

Docker Architecture

Docker uses a client-server architecture. The Docker client interacts with the Docker daemon that performs running, heavy lifting of building, and distribution of Docker containers.



Source: https://docs.docker.com/engine/docker-overview/#docker-architecture

Docker Architecture

Docker daemon

It accepts the Docker API requests and manages Docker objects, such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

Docker client

It is the primary path for Docker users to interact with the Docker application.

Docker Architecture

Docker registries

It stores Docker images. A Docker registry can be of classified into two categories:

Local Registry: It helps the user to pull the image.

Docker Trusted Registry: It is a feature of the Docker Enterprise that helps the user to pull the image and scan the image.

Docker objects

When the user uses Docker, in order to package the application and store it in isolated bundles, user creates and uses objects, such as images, containers, networks, volumes, and plugins.

Docker Ecosystem

Docker Ecosystem comprises various components listed below:

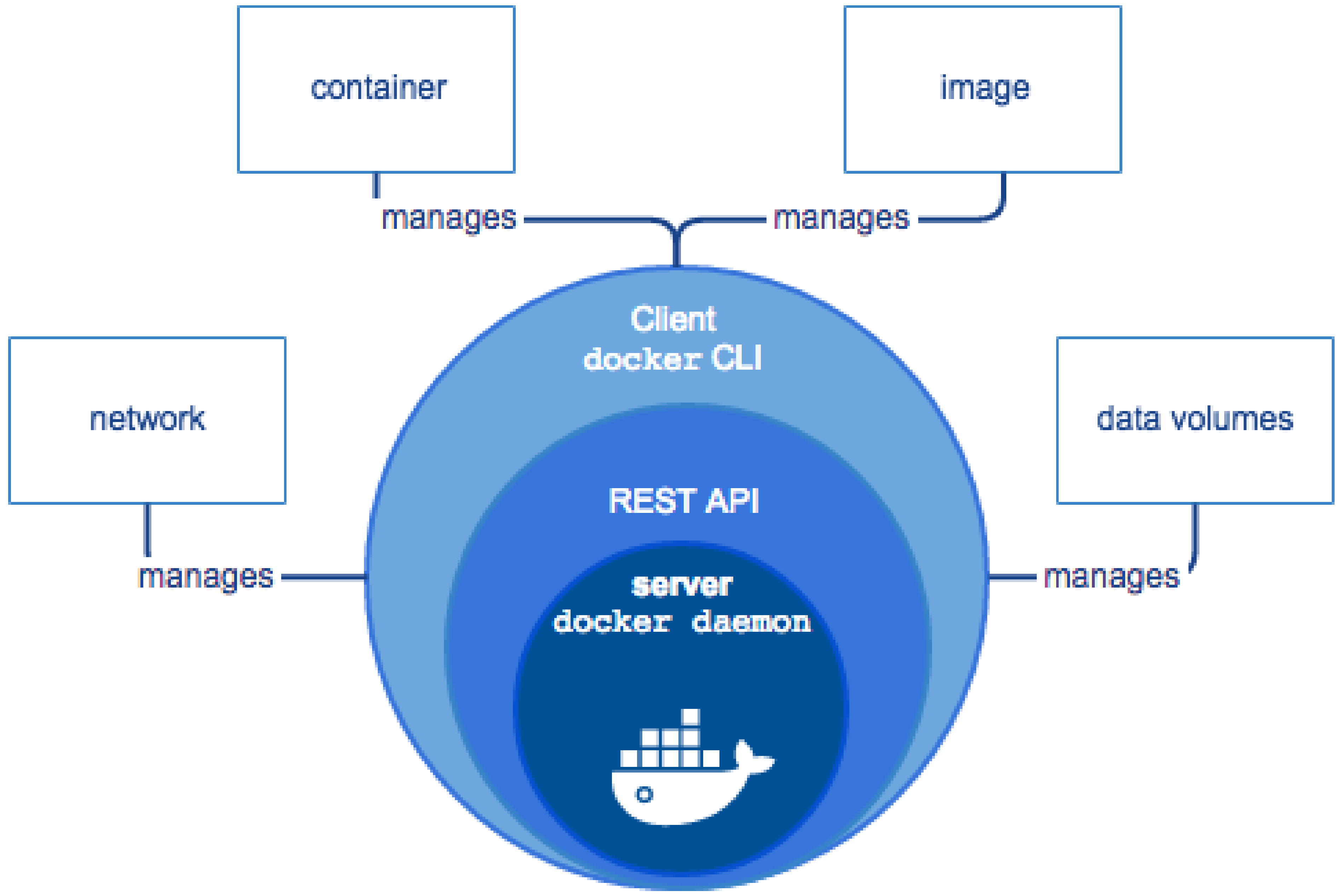
01	Docker Engine
02	Kinematic
03	Docker Machine and Swarm
04	Docker Compose
05	Docker Cloud
06	Data Center

Docker Engine

Docker Engine is a client-server application with these major components:

- **Server:** It is a type of long-running program called a daemon process (the dockerd command).
- **REST API:** It specifies the interfaces that programs can use to communicate with the daemon and instructs it on what to do next.
- **CLI:** It is a command line interface client that is used to write the docker commands.

Docker Engine



Source: https://docs.docker.com/engine/docker-overview/#docker-engine

Docker Engine

Docker Engine supports the tasks and workflows involved to build, ship, and run container-based applications. The engine creates a daemon process on the server side that hosts volumes of files, containers, networks, and storage.

Docker consists of:

- **The Docker Engine:** It is a lightweight and powerful open source containerization technology combined with a workflow for building and containerizing your applications.
- **Docker Hub:** It is a Software as a Service (SAAS) for sharing and managing your application stacks.

TECHNOLOGY

Docker Installation

Docker Installation

OS Requirements:

The user must have the 64-bit version of any one of the following Ubuntu versions:

- Disco 19.04
- Cosmic 18.10
- Bionic 18.04 (LTS)
- Xenial 16.04 (LTS)

Docker Community Edition Installation

Duration: 15 Min.

Problem Statement:

Install Docker Community Edition on Ubuntu.

Unassisted Practice: Guidelines

Steps to install Docker Community Edition:

1. Install Docker CE from Docker Repository.
2. Verify the correctly installed Docker Engine.

Docker Images and Containers

What Is Docker Image?

1 Read-only template combination of multiple layers

2 Comprises basic Operating System binaries

3 Stores Docker images on Docker Hub or private repositories

4 Hosts software applications, web containers, application servers, and even databases

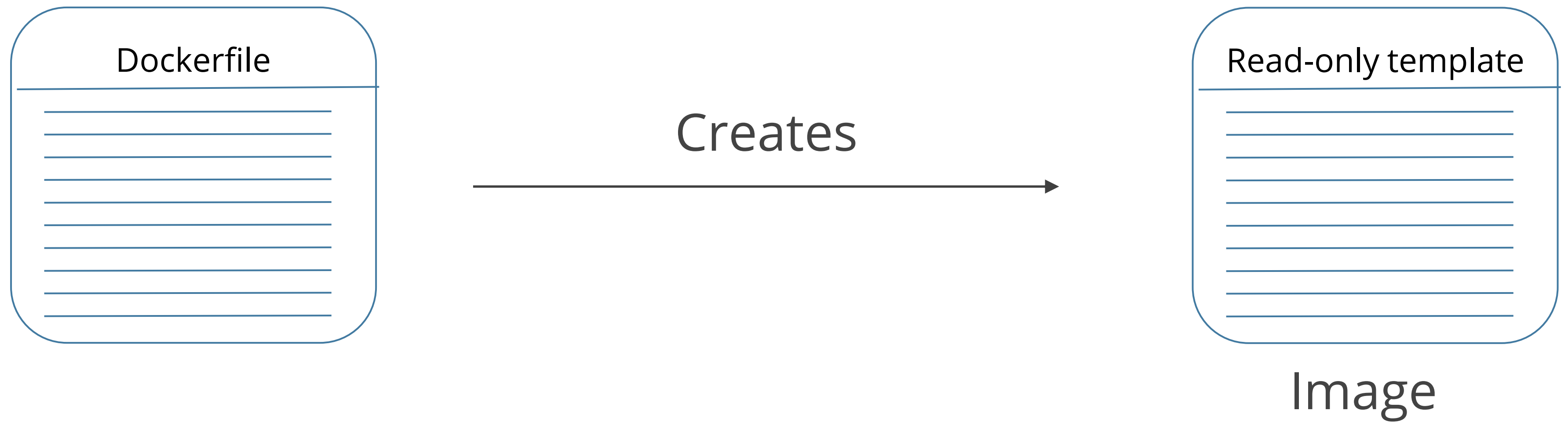
5 Exists for various open source tools in Docker Hub and its free to pull those images

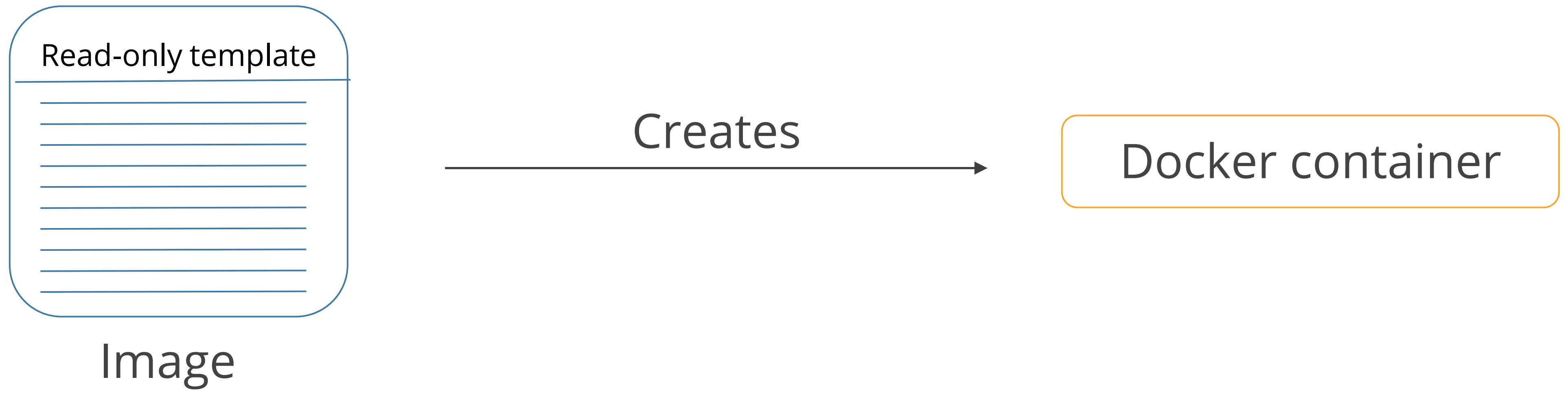
6 Images can be Built using custom Dockerfile

7 Smaller in size which helps to store and run Docker images easily

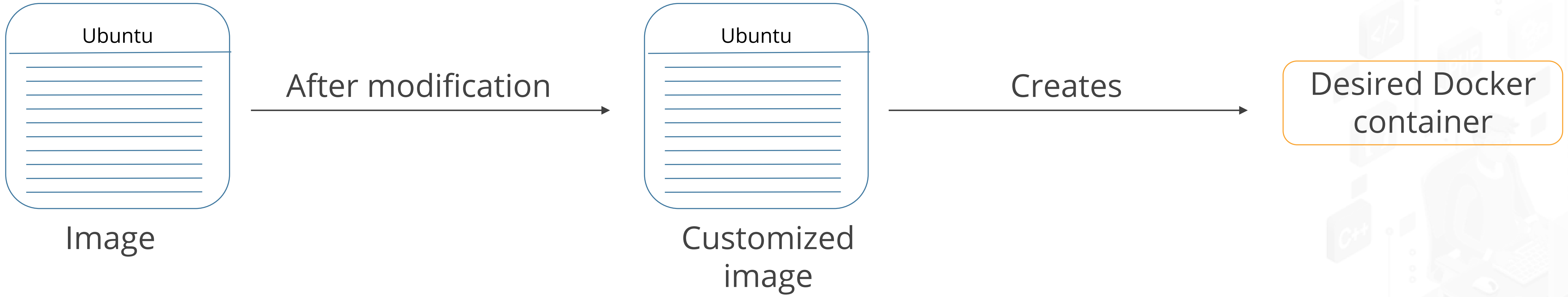
What Is Docker Image?

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

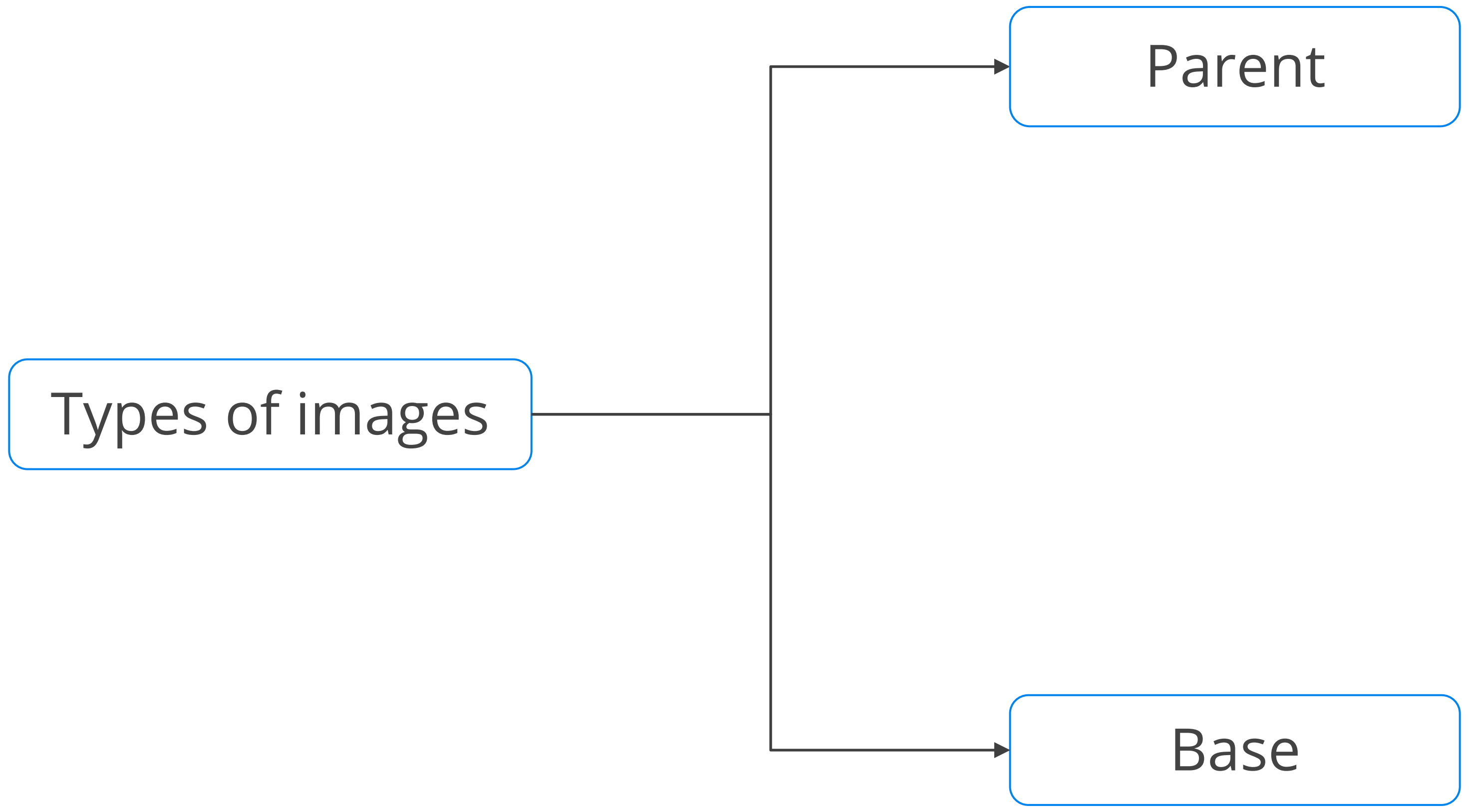




What Is Docker Image?



Types of Images



TECHNOLOGY

Execution on Docker Images

Execution on Docker Images

- The command to check all existing Docker images on Docker hosts is **docker images**.
- The command to pull a Docker image from either Docker Hub or private repository is **docker pull <image_name>:<tag>**.
- You can get the image name from Docker Hub or you can also search it using the command **docker search <image_name>**.
- Docker pull command will first check the image on local system. In case it is not found there, Docker image will be pulled from Docker Hub.

Execution on Docker Images

- You can remove Docker image using command given below:

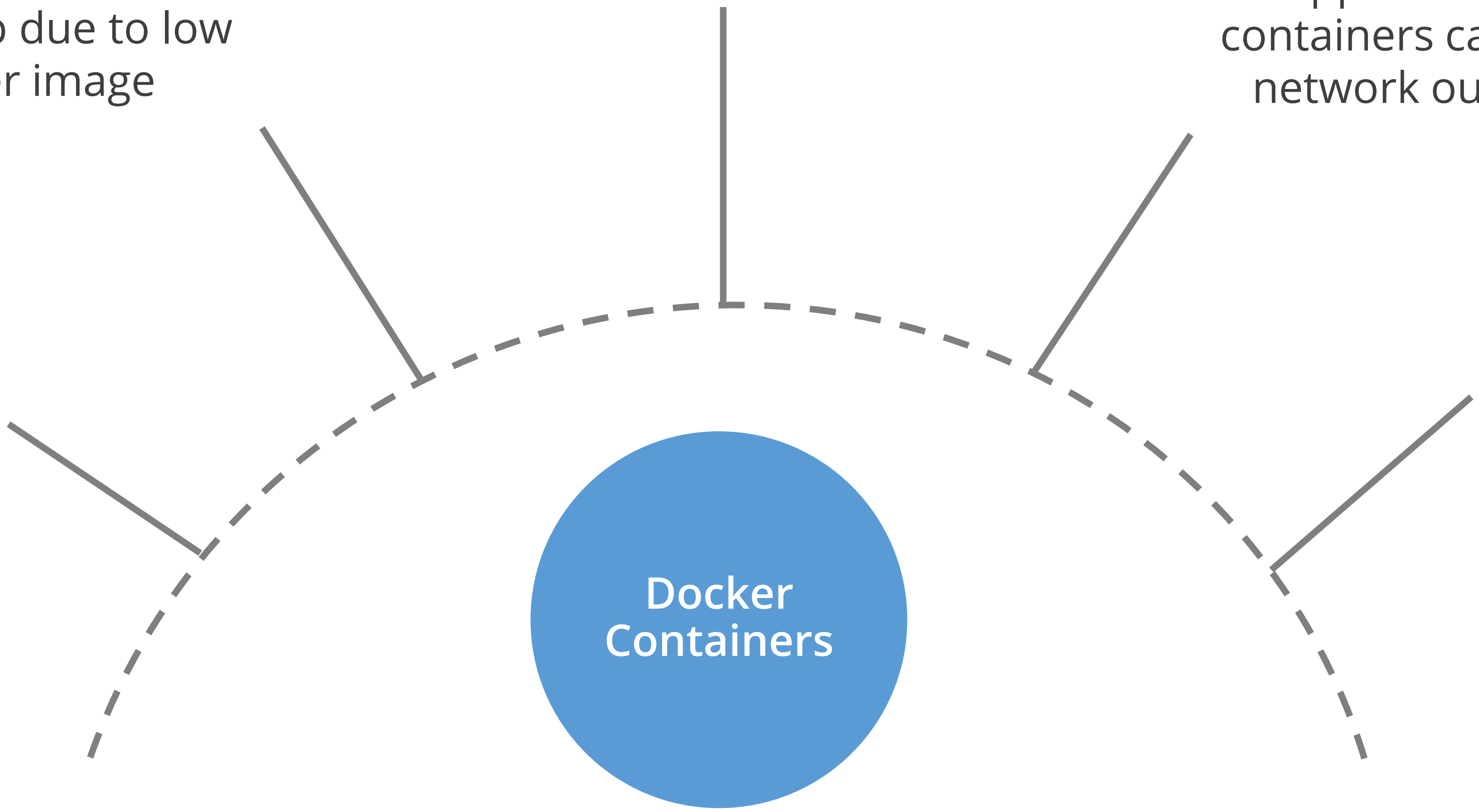
  docker rmi <image_name>:<tag>
- **Docker images** consumes local disk space, you have to continuously work on maintenance.
- You can also export Docker images to tar file so that they can be transferred from one server to another.

  docker save <image_name>:<tag> -o <tar_filename>
- Docker image can be imported with all layers from tar file using command below:

  docker load –I <tar_filename>
- All the previous details regarding image will be loaded automatically.

Docker Containers

An instance of Docker image is called Docker container. Docker containers host different types of applications inside them.

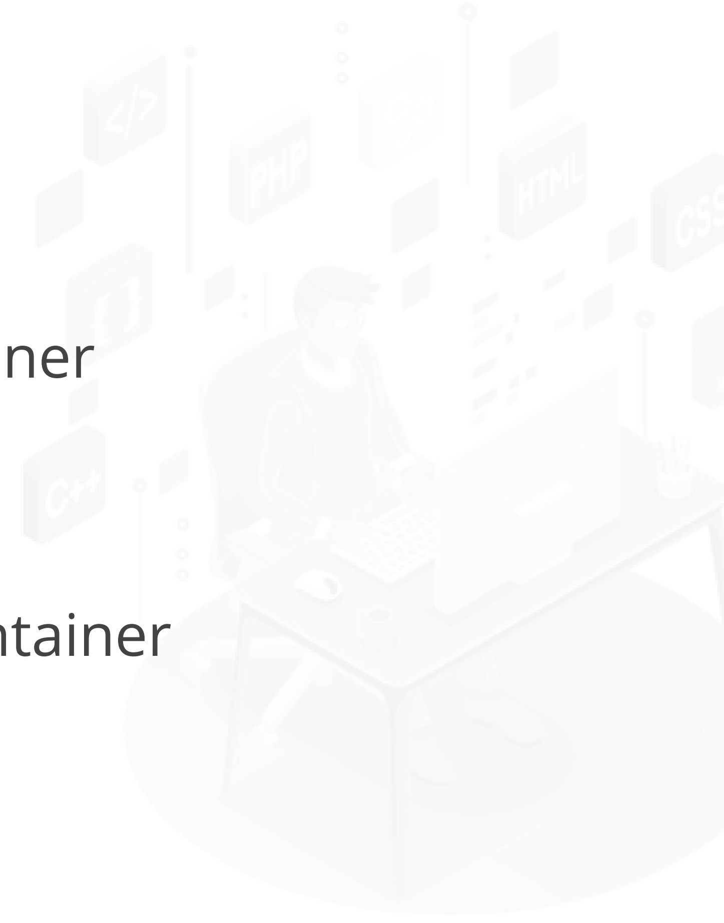


- Contain OS, network, and environment configurations
- Faster to boot up due to low size of Docker image
- Can host one or multiple applications at once
- Applications hosted inside containers can be accessed over network outside Docker Host
- Can be stopped, restarted, and killed

Container: Overview

Docker API and CLI help in:

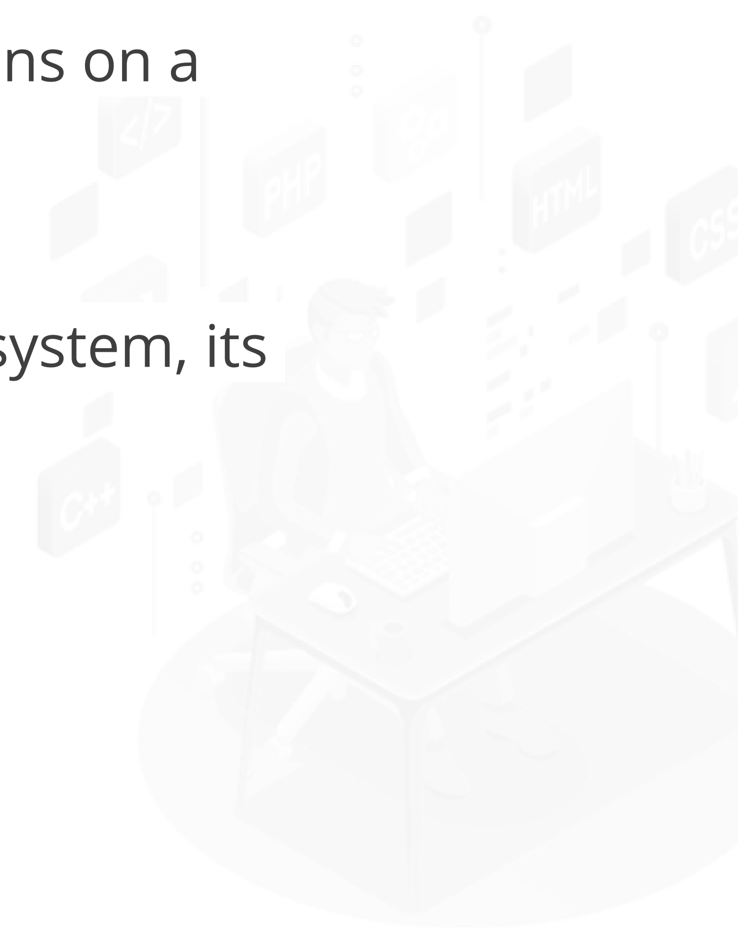
- Stopping a container
- Starting a container
- Moving a container
- Creating a container
- Deleting a container



TECHNOLOGY

Manipulating Container with Docker Client

Docker Run in Detail



- Docker runs processes in isolated containers. A container is a process which runs on a host. The host may be local or remote.
- When an operator executes docker run, the container process has its own file system, its own networking, and its own isolated process tree separate from the host.

Docker Run in Detail

The basic docker run command takes this form:

```
$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
```

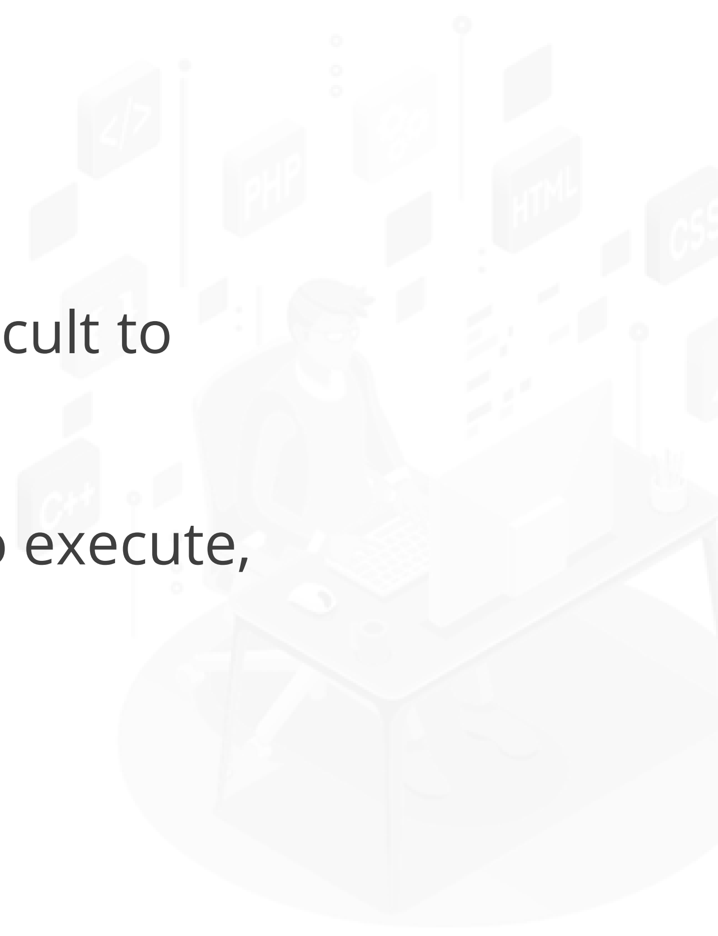
The docker run command must specify an *Image* to derive the container from.

An image developer can define image defaults related to:

- Detached or foreground running
- Container identification
- Network settings
- Runtime constraints on CPU and memory

Overriding Default Commands

- Every Docker image has an entry point.
- It can be the default one or the one specified by the ENTRYPOINT instruction.
- It's a default command to be executed at runtime.
- It specifies what executable to run when the container starts, but it is more difficult to override.
- The ENTRYPOINT makes a running container more like a complete command to execute, which takes parameters from the COMMAND value.



Overriding Default Commands

Syntax for Overriding Default command at entry point is:

```
docker run -it --entrypoint /bin/bash Ubuntu
```

Let's take at look of the MongoDB Dockerfile, which has the ENTRYPOINT defined:

```
FROM dockerfile/ubuntu
RUN mkdir -p /data/db /data/configdb \
&&chown -R mongodb:mongodb /data/db /data/configdb
VOLUME /data/db /data/configdb
COPY docker-entrypoint.sh /entrypoint.sh
ENTRYPOINT ["/entrypoint.sh"]
EXPOSE 27017
CMD ["mongod"]
```

Overriding Default Commands

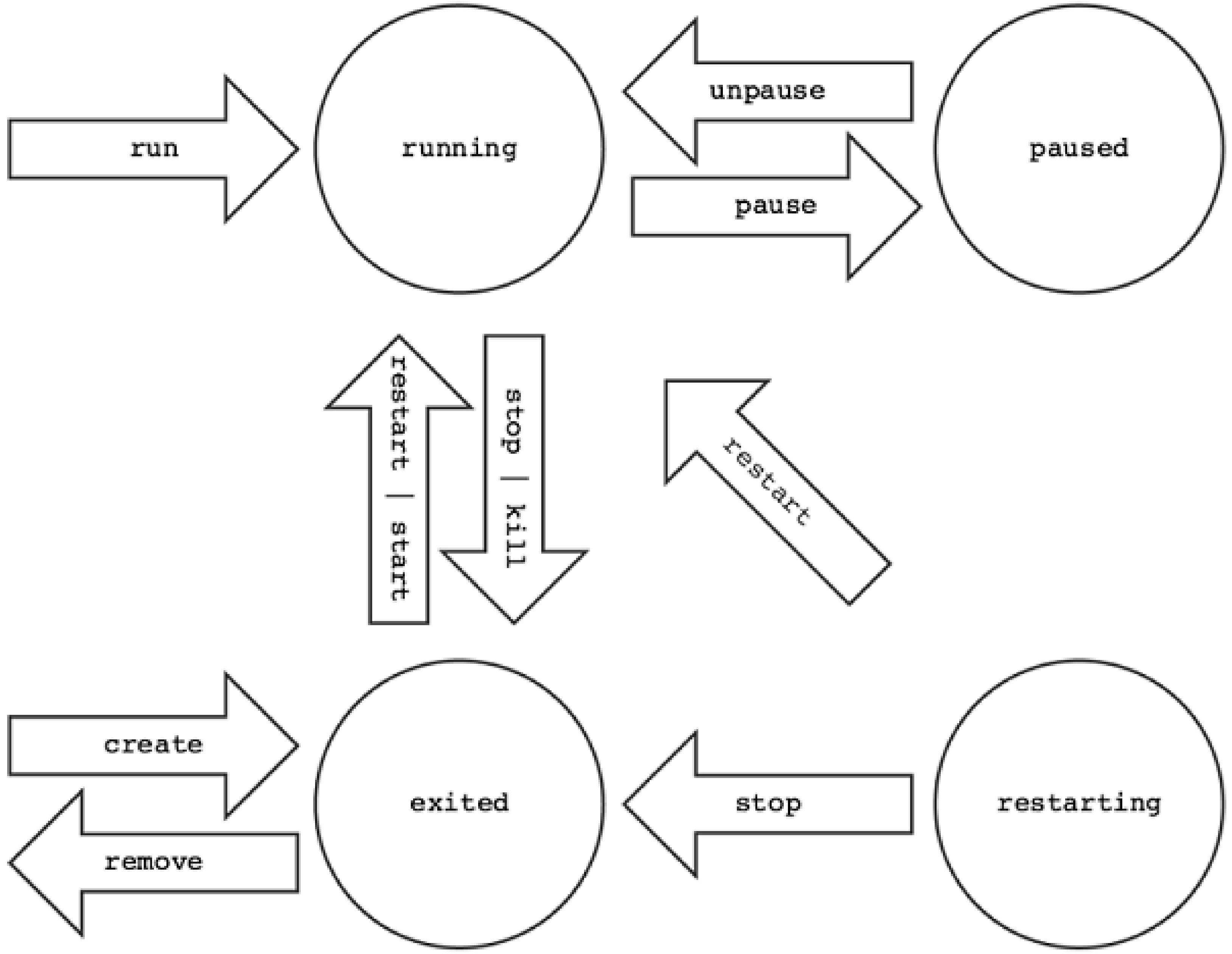
- In the previous example, it has a custom entry point defined.
- It will be an **entrypoint.sh** shell script.
- If you run this container, the main process of the container will come from this script.
- We can override this entry point using the command:

```
docker run -it --entrypoint /bin/bash mongo
```

- On executing this command, the ENTRYPOINT from the Dockerfile will be overridden by the one provided during the docker run command.

Container Lifecycle

Below are the various phases involved in container lifecycle:



docker exec

The syntax to execute a command in a running container is as shown below:

```
docker exec [OPTIONS] CONTAINER COMMAND [ARG…]
```

- The docker exec command runs a new command in a running container.
- The command started using docker exec only runs while the container's primary process (PID 1) is running, and it is not restarted if the container is restarted.
- COMMAND will run in the default directory of the container.
- If the underlying image has a custom directory specified with the WORKDIR directive in its Dockerfile, this will be used instead.

Start a Docker Shell

The syntax to execute docker start command is as shown below:

```
docker start [OPTIONS] CONTAINER  [CONTAINER…]
```

This command is used to start all the stopped containers.

For example:

```
$ docker start my_example_ container
```

Performing CRUD Operation on Containers

Problem Statement:

Duration: 45 Min.

You have been asked to perform CRUD operations on containers.

Assisted Practice: Guidelines

Steps to Perform:

1. Pull a Docker image.
2. Create a new container.
3. Stopp the container.
4. List all the containers.
5. Delete the container.
6. Remove the image.

Custom Image Through Docker Server

Parent Image and Base Image

- Most Dockerfiles start from a parent image.
- If you need to completely control the contents of your image, you might need to create a base image instead.
- A parent image is the image that your image is based on. It refers to the contents of the FROM directive in the Dockerfile.
- Each subsequent declaration in the Dockerfile modifies this parent image.
- Most Dockerfiles start from a parent image, rather than a base image.
- A base image has FROM scratch in its Dockerfile.

Creating a Base Image

Create a base image for Docker using tar as shown below:

```
$ sudo tar -C xenial -c . | docker import - xenial

a29c15f1bf7a

$ docker run xenial cat /etc/lsb-release

DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=16.04
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION="Ubuntu 16.04 LTS"
```

Creating a Base Image

Create a simple image from scratch as shown below:

```
FROM scratch
ADD hello /
CMD ["/hello"]
```

To build the image, simply use the below command:

```
docker build --tag hello .
```

Creating a Docker Image

Duration: 15 Min.

Problem Statement: You have been asked to create a docker image using Dockerfile

Assisted Practice: Guidelines

Steps to Perform :

1. Create the Dockerfile
2. Execute the Dockerfile.

Build Process

Docker API and CLI helps in:

- Install Docker
- Build first Docker image
- Tag a Dockerfile
- Run a Docker image
- Push Docker image to the repository

Tagging a Docker Image

- When you have many images, it becomes difficult to know which image is what.
- Docker provides a way to tag your images with friendly names of your choice. This is known as tagging.

```
$ docker build -t yourusername/repository-name .
```

Example:

```
$ docker build -t yourusername/example-node-app
```

- If you run the command above, you should have your image tagged already. If you run the docker images, it will display the name that you have tagged.

TECHNOLOGY

Docker Compose with Multiple Local Containers

Introduction to Docker Compose

- Compose is a tool for defining and running multi-container Docker applications.
- With Compose, you use a YAML file to configure your application’s services.
- Then, with a single command, you create and start all the services from your configuration.
- Compose works in all environments: production, staging, development, testing, as well as CI workflows.

Docker Compose .yml file

The Docker compose .yml file looks as shown below with the following components:

```yaml
version: "3.9"  # optional since v1.27.0
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

Flow of Docker Compose

- Define your app's environment with a Dockerfile so it can be reproduced anywhere.
- Define the services that make up your app in **docker-compose.yml** so that they can run together in an isolated environment.
- Run docker compose up and the Docker compose command starts and runs your entire app. You can alternatively run docker-compose up using the docker-compose binary.

Networking in Compose

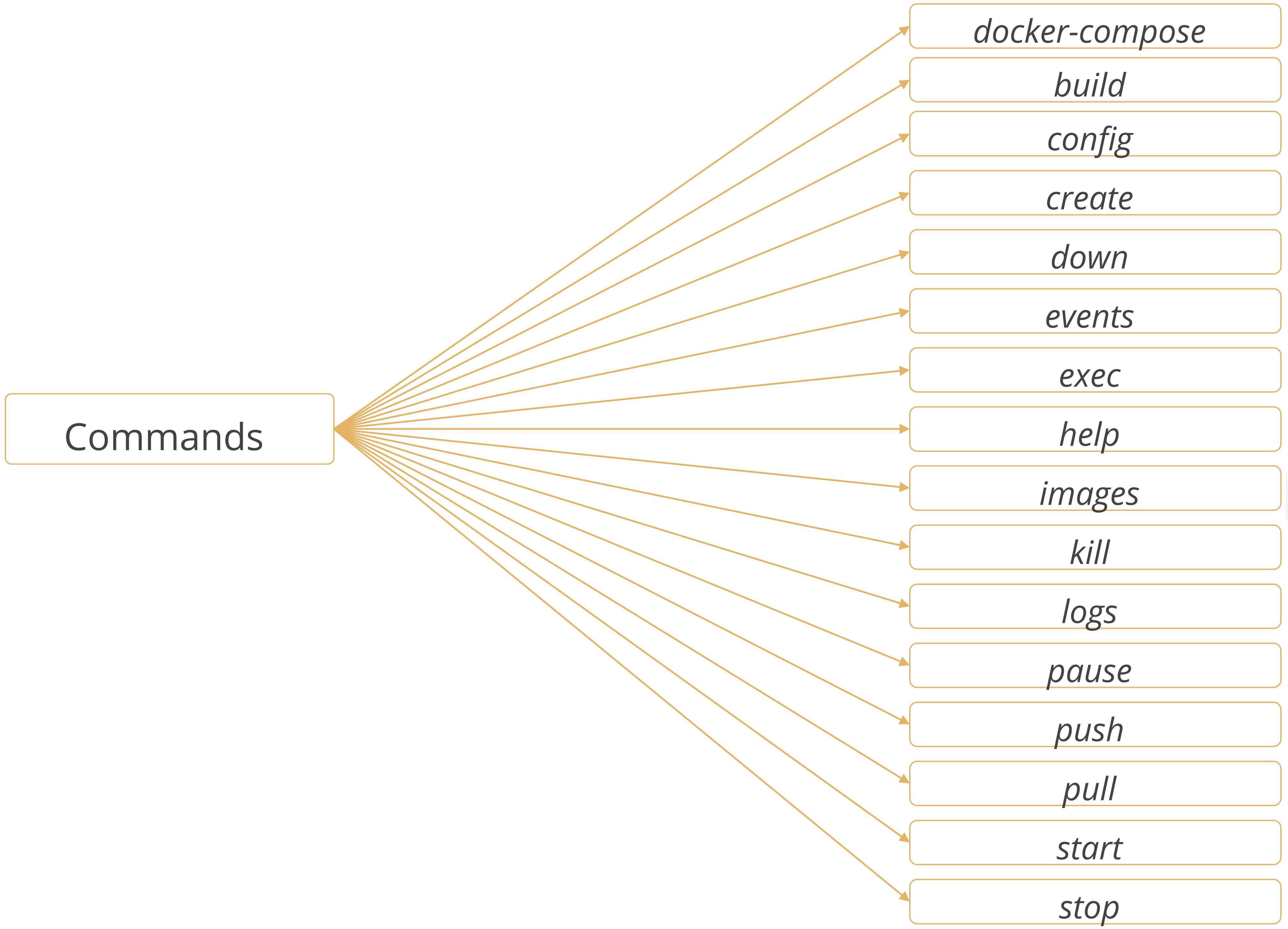
- By default, compose sets up a single network for your app.
- Each container for a service joins the default network and is both reachable by other containers on that network.
- It is discoverable by them at a hostname identical to the container name.

Networking in Compose

Suppose an app is in a directory called **myexampleapp**, and the **docker-compose.yml** looks like this:

```
version: "3.9"
services:
  web:
    build: .
    ports:
      - "8000:8000"
  db:
    image: postgres
    ports:
      - "8001:5432"
```

Docker Compose Commands



Docker Compose Setup

Duration: 15 Min.

Problem Statement: You have been asked to use the docker compose to manage a container.

Assisted Practice: Guidelines

Steps to Perform:

1. Set up docker-compose
2. Create a docker-compose file

Docker Registry

Duration: 15 Min.

Problem Statement: You have been asked to run a docker registry from official registry image.

Assisted Practice: Guidelines

Steps to Perform:

1. Pull a Linux container
2. Push the image to the local repository
3. Run a new image

Docker Networking with SSHs

Duration: 15 Min.

Problem Statement: You have been asked to use the docker network and its configuration.

Assisted Practice: Guidelines

Steps to Perform:

1. Create a container and commit it
2. Create a bridge network and find its IP address
3. Connect the network from another SSH server

Automatic Container Restarts

The syntax for docker compose restart is:

restart [options] [SERVICE...]

- It restarts all stopped and running services.
- If you make changes to your docker-compose.yml configuration, these changes are not reflected after running this command.
- For example, changes to environment variables (which are added after a container is built, but before the container's command is executed) are not updated after restarting.

TECHNOLOGY

Docker Hub

Getting an Image into Docker Hub

1. Log in on **https://hub.docker.com/**

1. Click on *Create Repository*

1. Choose a name (e.g. my_hub), add a description for your repository, and click *Create*

1. Log in to the Docker Hub from the command line

```
docker login --username=yourhubusername --email=youremail@company.com
```

Pushing an Image into Docker Hub

1. Check the image ID using

   ```
   docker images
   ```

1. Push your image to the repository you created

   ```
   docker push yourhubusername/verse_gapminder
   ```

Saving and Loading Images

To save a Docker image after you have pulled, committed or built it, you use the docker save command.

For example, let’s save a local copy of the my_hubr docker image created:

```
docker save my_hub > my_hub.tar
```

If we want to load that Docker container from the archived tar file in the future, we can use the docker load command:

```
docker load --input my_hub.tar
```

Key Takeaways

- Docker is a platform for developers and sysadmins to develop, ship, and run applications by using containers.
- Docker Engine supports the tasks and workflows involved to build, ship, and run container-based applications.
- Docker Compose is a tool for defining and running multi-container Docker applications.
- Docker Hub is a service provided by Docker for finding and sharing container images with the team.

Lesson-End Project

Containerizing Legacy Docker Application

Project Agenda: As a DevOps engineer, you are required to containerize the legacy application system using Docker..

Description: Your team has been asked to sync the Django application and the database so that the information can be stored and accessed again, on demand. This is a legacy application with two components: a Django application and a Postgres database. Now, your project manager asks you to containerize the legacy system using Docker. So, you are required to create Docker images for these components using the Dockerfile and connect them using the docker-compose file.

LESSON-END PROJECT