Docker

Introduction

Docker

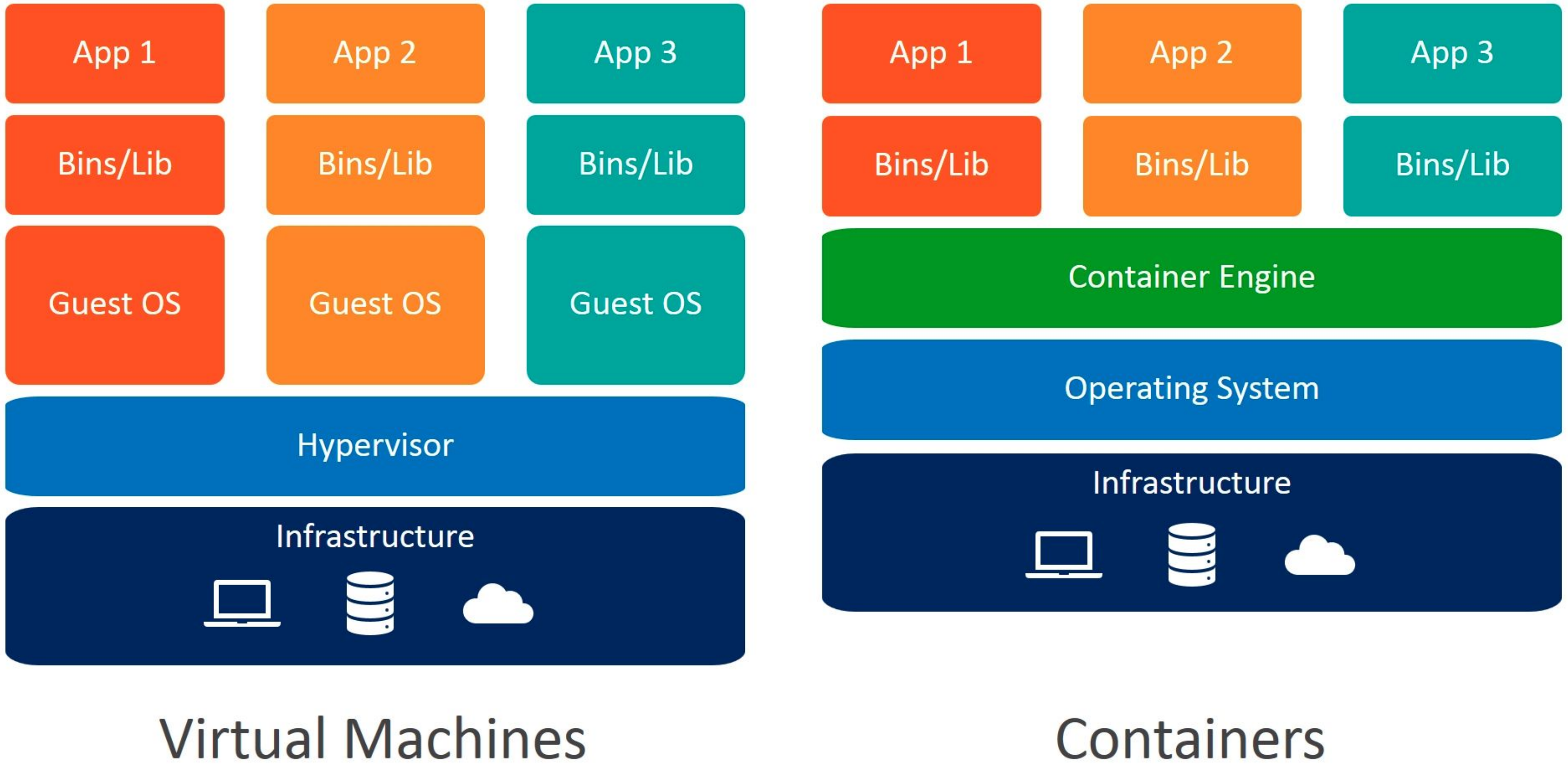
- **Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers. The service has both free and premium tiers. The software that hosts the containers is called Docker Engine.**
- **Docker can package an application and its dependencies in a virtual container that can run on any Linux, Windows, or macOS computer. This enables the application to run in a variety of locations, such as on-premises, in public (see decentralized computing, distributed computing, and cloud computing) or private cloud.[10] When running on Linux,**
- **Docker uses the resource isolation features of the Linux kernel (such as cgroups and kernel namespaces) and a union-capable file system (such as OverlayFS)[11] to allow containers to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines.[12] Docker on macOS uses a Linux virtual machine to run the containers.**

What is a Container

Simply put, a container is a sandboxed process on your machine that is isolated from all other processes on the host machine. That isolation leverages kernel namespaces and cgroups, features that have been in Linux for a long time. Docker has worked to make these capabilities approachable and easy to use. To summarize, a container:

- **is a runnable instance of an image. You can create, start, stop, move, or delete a container using the DockerAPI or CLI.**
- **can be run on local machines, virtual machines or deployed to the cloud.**
- **is portable (can be run on any OS).**
- **is isolated from other containers and runs its own software, binaries, and configurations.**

Container vs VMs

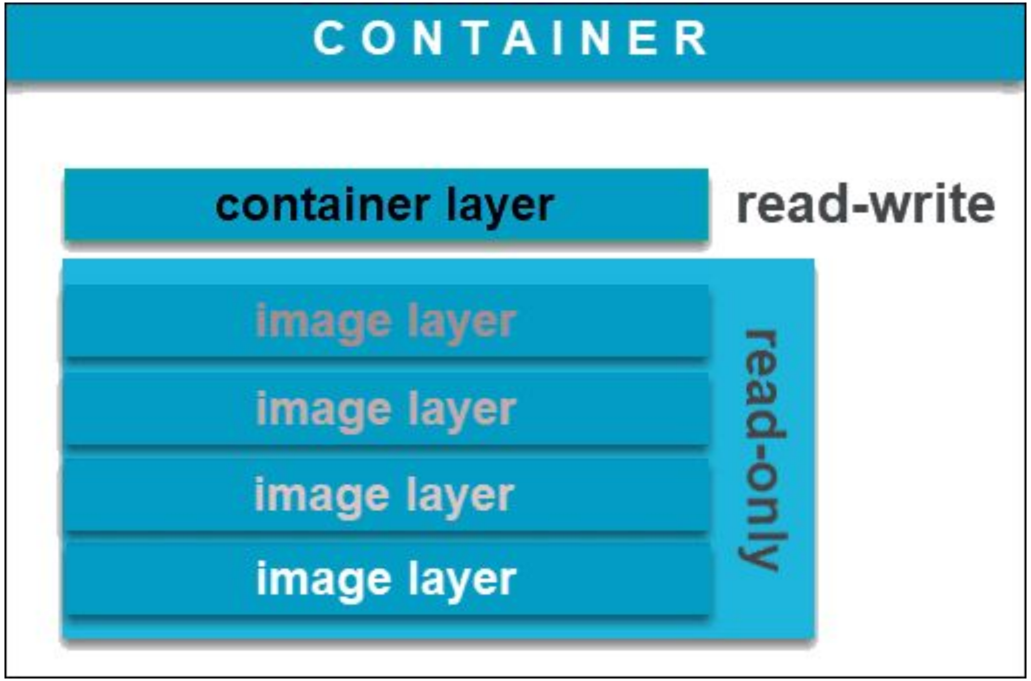


Docker Image

Images can exist without containers, whereas a container needs to run an image to exist. Therefore, containers are dependent on images and use them to construct a run-time environment and run an application. A container is a running Image

A **Docker image** is an immutable (unchangeable) file that contains the source code, libraries, dependencies, tools, and other files needed for an application to run.

You can create an unlimited number of Docker images from one **image base**. Each time you change the initial state of an image and save the existing state, you create a new template with an additional layer on top of it.



Dockerfile

- A Dockerfile is a script that contains instructions for building a customized docker image. Each instruction in a Dockerfile creates a new layer in the image, and the final image is composed of all the layers stacked on top of each other.
- It includes instructions for installing dependencies, copying files, setting environment variables, and configuring the container.
- Dockerfile uses a simple, easy-to-read syntax that can be created and edited with any text editor. Once a Dockerfile has been created, it can be used to build an image using the docker build command.
-

Install Docker on Ubuntu

1. sudo apt-get update
2. sudo apt-get install ca-certificates curl gnupg lsb-release
3. sudo mkdir -p /etc/apt/keyrings
4. curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
5.
```
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
```
6. sudo apt-get update
7. sudo chmod a+r /etc/apt/keyrings/docker.gpg
8. sudo apt-get update

Install Docker on Ubuntu

1. sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
2. sudo docker run hello-world

Docker Commands

Docker build —-- dockerfile to image

Docker run   —--image to container and run it

Docker ps

Docker create —--- image to container but does not run it

Docker kill

Docker push

Docker pull

Docker rm

Docker Pull

- Most of your images will be created on top of a base image from the Docker Hub registry.
- Docker Hub contains many pre-built images that you can pull and try without needing to define and configure your own.
- To download a particular image, or set of images (i.e., a repository), use docker pull.

```
$ sudo docker pull hello-world
```

Docker Build

Command	Explanation
docker build	Builds an image from a Dockerfile in the current directory
docker build https://github.com/docker/rootfs.git#container:docker	Builds an image from a remote GIT repository
docker build -t imagename/tag	Builds and tags an image for easier tracking
docker build https://yourserver/file.tar.gz	Builds an image from a remote tar archive
docker build -t image:1.0 -<<EOFFROM busyboxRUN echo “hello world”EOF	Builds an image via a Dockerfile that is passed through STDIN

Docker run

The `docker run` command runs a command in a new container, pulling the image if needed and starting the container.

You can restart a stopped container with all its previous changes intact using `docker start`. Use `docker ps -a` to view a list of all containers, including those that are stopped.

```
$ docker run --name test -it debian
```

```
$ docker run --name hello hello-world
```

-i option keeps the STDIN interactive and -t option open a pseudo terminal for the container.

Docker ps

The '*docker ps*' is a Docker *command* to list the running containers by default; however, we can use different flags to get the list of others.

$ docker ps

`--all` , `-a`		Show all containers (default shows just running)
`--filter` , `-f`		Filter output based on conditions provided
`--format`		Format output using a custom template: ‘table’: Print output in table format with column headers (default) ‘table TEMPLATE’: Print output in table format using the given Go template ‘json’: Print in JSON format ‘TEMPLATE’: Print output using the given Go template. Refer to https://docs.docker.com/go/formatting/ for more information about formatting output with templates
`--last` , `-n`	-1	Show n last created containers (includes all states)
`--latest` , `-l`		Show the latest created container (includes all states)
`--no-trunc`		Don’t truncate output
`--quiet` , `-q`		Only display container IDs
`--size` , `-s`		Display total file sizes

Run an Existing Container Image

Docker run command is used to run a container

```
docker run --name mynginx1 -p 80:80 -d nginx
```

The above command starts an nginx container which can be verified using

```
docker ps
```

Docker ps command lists the running containers on the docker engine

Docker Hub

Docker Hub is a hosted repository service provided by Docker for finding and sharing container images with your team. Key features include:

- **Private Repositories: Push and pull container images**
- **Automated Builds: Automatically build container images from GitHub and Bitbucket and push them to Docker Hub**
- **Teams & Organizations: Manage access to private repositories**
- **Official Images: Pull and use high-quality container images provided by Docker**
- **Publisher Images: Pull and use high-quality container images provided by external vendors. Certified images also include support and guarantee compatibility with Docker Enterprise**
- **Webhooks: Trigger actions after a successful push to a repository to integrate Docker Hub with other services**

Create a Docker Image from App

Step 1. Mkdir java-app

Step 2: Create a .java file

Step 3: Create a .java file save it as Hello.java

Step 4: Create a dockerfile

```
$touch dockerfile
```

Step 4: build the dockerfile inside java-app folder

```
sudo docker build -t java-app .
```

Step 4: run the docker image

```
sudo docker run java-app
```

```
Class Hello

{

        public static void main(String[] args)

        {

                System.out.println("Hello World");

        }

}
```

```
FROM openjdk:17
COPY /home/ubuntu/java-app /tmp
WORKDIR /tmp
RUN javac Hello.java
CMD ["java", "Hello"]
```

Upload Your Custom Image to the Hub

Upload the image created in the previous slide and add it to docker hub. Create another instance and pull and run the image in the new instance.

Assignments

1. Pull and run the hello-world image
2. Pull and run the nginx image, check if the webpage is actively hosted.
3. Pull and run mysql container from hub. Can you log into the tty provided using -it option.
4. Pull the nginx image from docker hub, rename the image to mynginx and upload it to docker hub.

Containerize an Application

https://docs.docker.com/get-started/02_our_app/

Assignment

- **Create a Ec2 instance and then install docker in it. Run the getting started app and create a container out of. Upload your container image to docker hub. Can you run the container created by you in any other system that runs docker engine.**
- **Write a bash script to create VPC and a subnet, fullfill all network requirements (route tables, routes and IG). Pass another script to the instance that installs and configures docker and install msql. Create and display a database in it.**
- **Create 2 instances in two different subnets but within the same VPC, show that instances within a VPC can communicate with each other(ssh, ping).**
-

Adding a Container to Docker Hub

```
# Build the image
docker build -t my-app .

# Tag the image (replace with your username and repository name)
docker tag my-app your_dockerhub_username/my-app:latest

# Login to Docker Hub
docker login

# Push the image
docker push your_dockerhub_username/my-app:latest

# Verify on Docker Hub
```

Assignment

- Use a bash script to automate configuration of an EC2 instance. Apache is the alternative of Nginx and is another open source web server. Write a dockerfile that pulls the latest apache image and runs a container. Display the welcome page of Apache along with your name.
- Using docker configure an EC2 instance. Write a Java Application that takes user input from a form for username. Use this input to create a database with his/her name.
- Write a bash script to create 2 instances in 2 different public subnets of the same VPC. Configure the to-do app in one of the instances and the persistent volume for the app in another instance.