

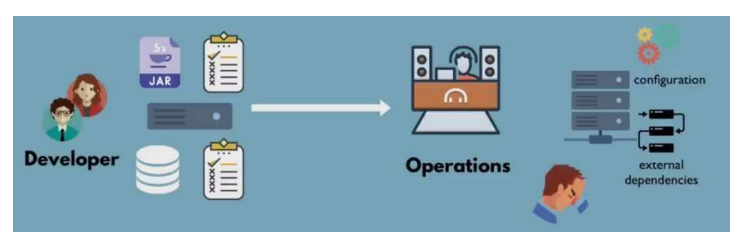
**What is Application Deployment?**

1. The process of deploying an application over executable environment for executing the application**.**
2. Application deployment example Refer Traditional deployment in the following section.
3. Application deployment before containers.

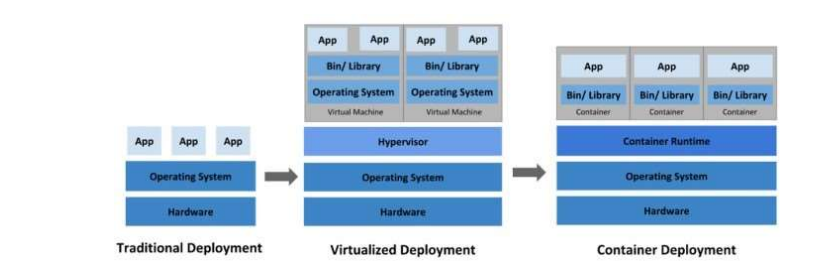
⎫ Before the containers, in the traditional deployment process, deployment team produce artifact with a set of instructions which include how to deploy these artifact and database services and maybe some other services which needed for the application.

⎫ The development team handover those artifacts to the production team to set up the environment to deploy the application.

⎫ In that kind of situation, first, we have to configure the server environment, and other disadvantage is dependency conflicts can occur in the traditional deployment process.



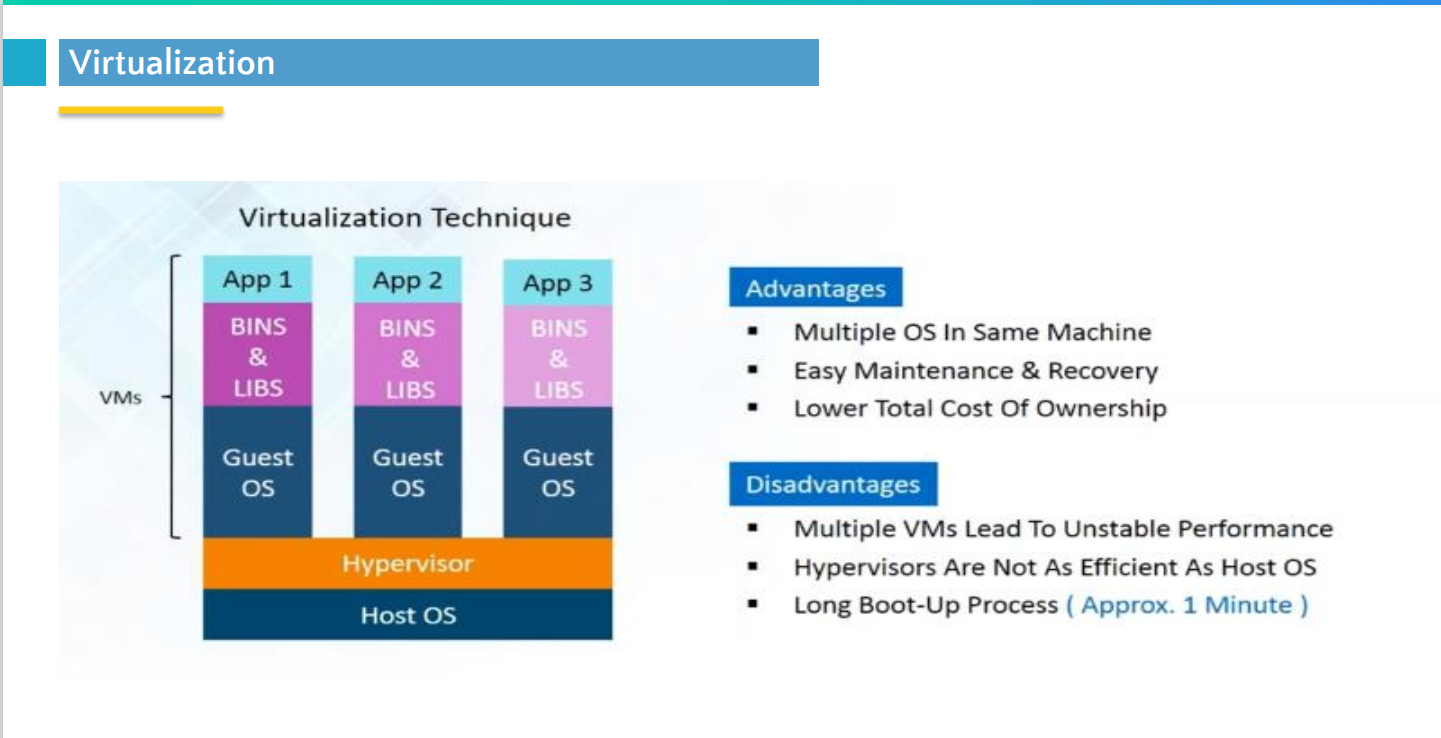
1. Application deployment transition over the period of time.



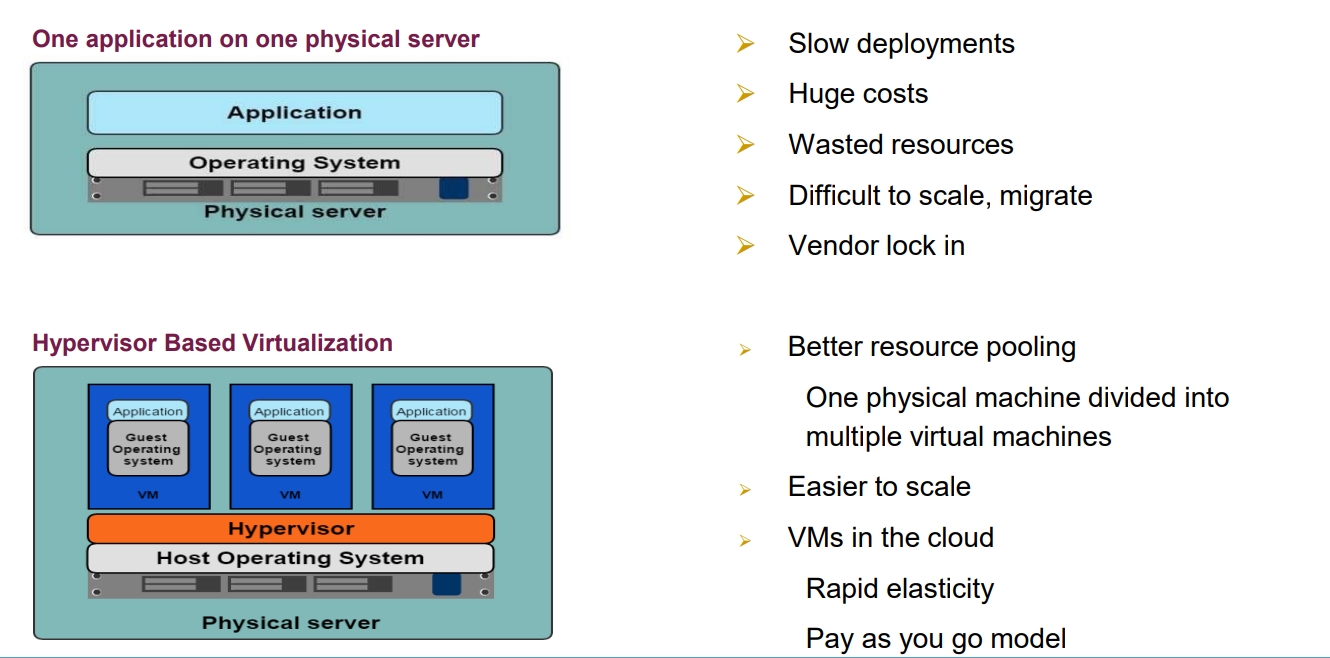
What is Virtualization?

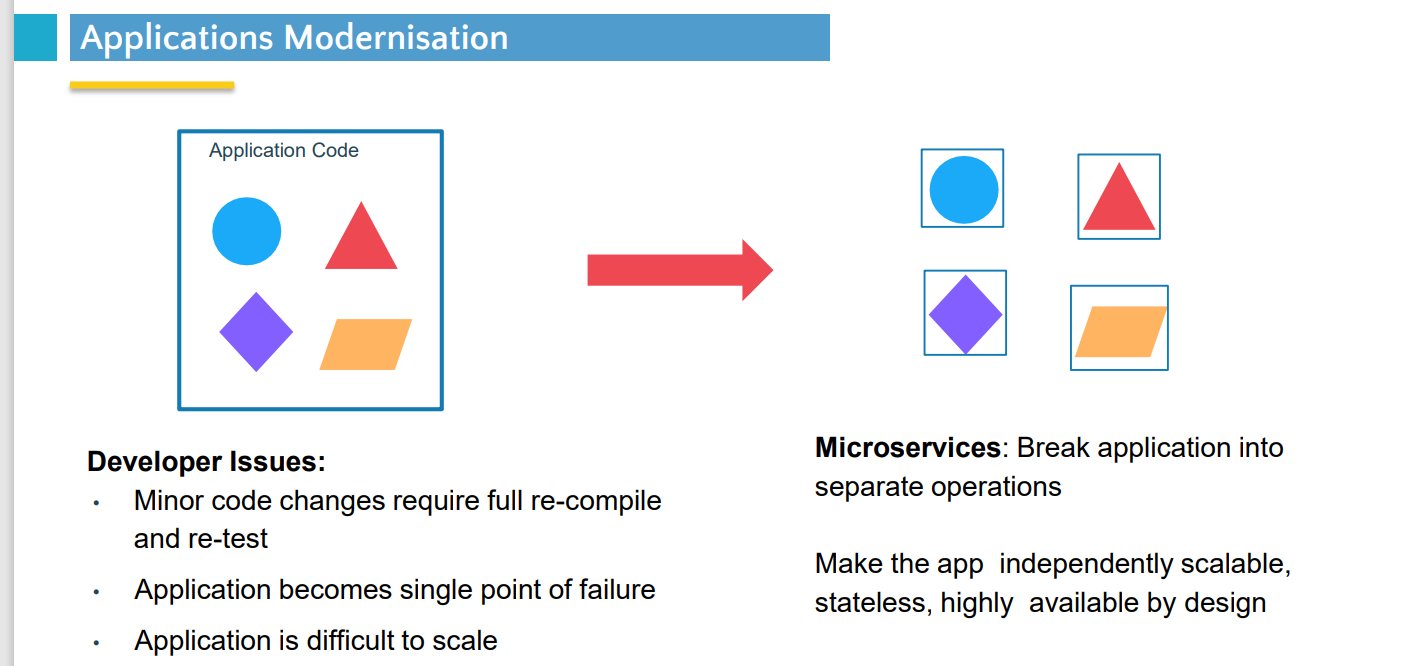
Virtualization is technology that you can use to create virtual representations of servers, storage, networks, and other physical machines. Virtual software mimics the functions of physical hardware to run multiple virtual machines simultaneously on a single physical machine.

Businesses use virtualization to use their hardware resources efficiently and get greater returns from their investment. It also powers cloud computing services that help organizations manage infrastructure more efficiently.



* Each VM still requires CPU allocation, Storage, RAM, Guest OS.
* The more VMs you run, the more resources you need.
* Application portability not guaranteed.



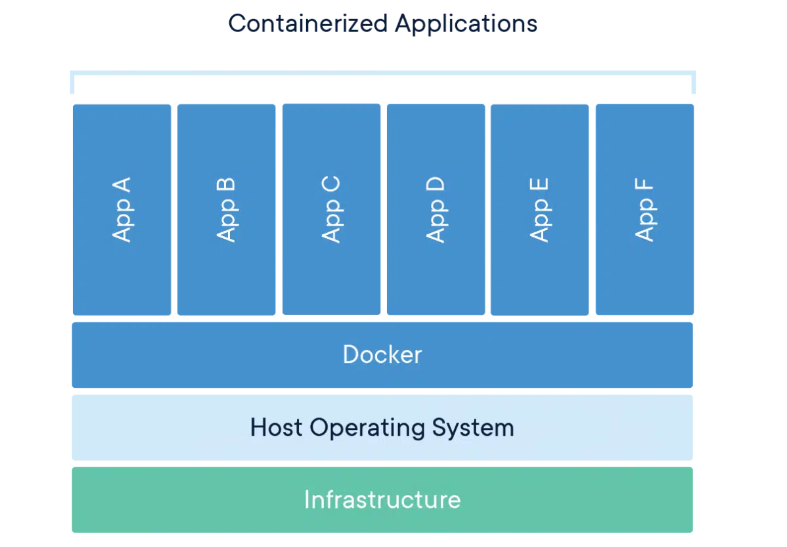


**What is Container:**

**A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.**

**A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.**

**Container images become containers at runtime and in the case of Docker containers – images become containers when they run on**[**Docker Engine**](https://www.docker.com/products/container-runtime)**.**

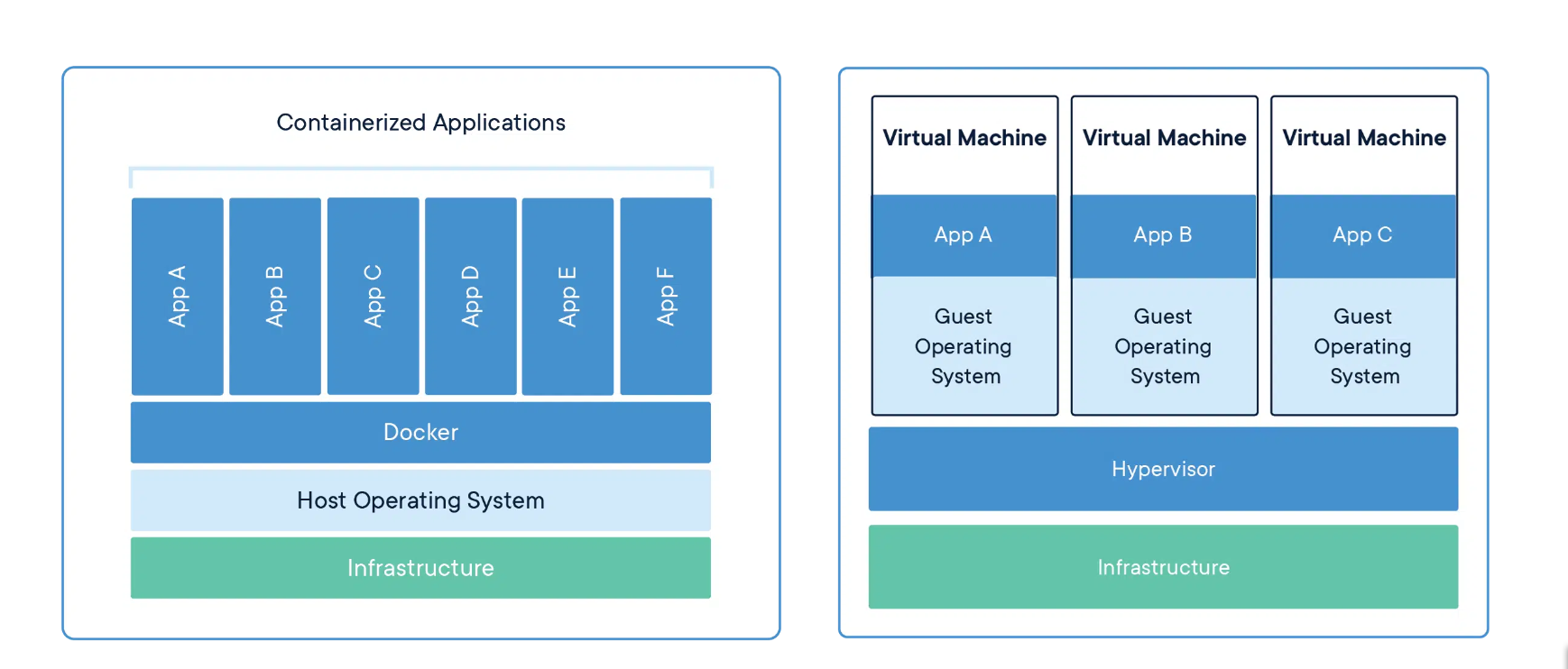


**Docker containers that run on Docker Engine:**

* **Standard: Docker created the industry standard for containers, so they could be portable anywhere**
* **Lightweight: Containers share the machine’s OS system kernel and therefore do not require an OS per application, driving higher server efficiencies and reducing server and licensing costs**
* **Secure: Applications are safer in containers and Docker provides the strongest default isolation capabilities in the industry**

**Comparing Containers and Virtual Machines:**

**Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware. Containers are more portable and efficient.**

****

# **CONTAINERS :**

# **Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), can handle more applications and require fewer VMs and Operating systems.**

### **VIRTUAL MACHINES :**

**Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, the application, necessary binaries and libraries – taking up tens of GBs. VMs can also be slow to boot.**

# **Docker overview:**

* **Docker is an open platform for developing, shipping, and running applications.**
* **Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications.**
* **By taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production.**

# **[The Docker platform](https://docs.docker.com/get-started/overview/" \l "the-docker-platform):**

* **Docker provides the ability to package and run an application in a loosely isolated environment called a container.**
* **The isolation and security lets you to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you don't need to rely on what's installed on the host.**
* **You can share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.**

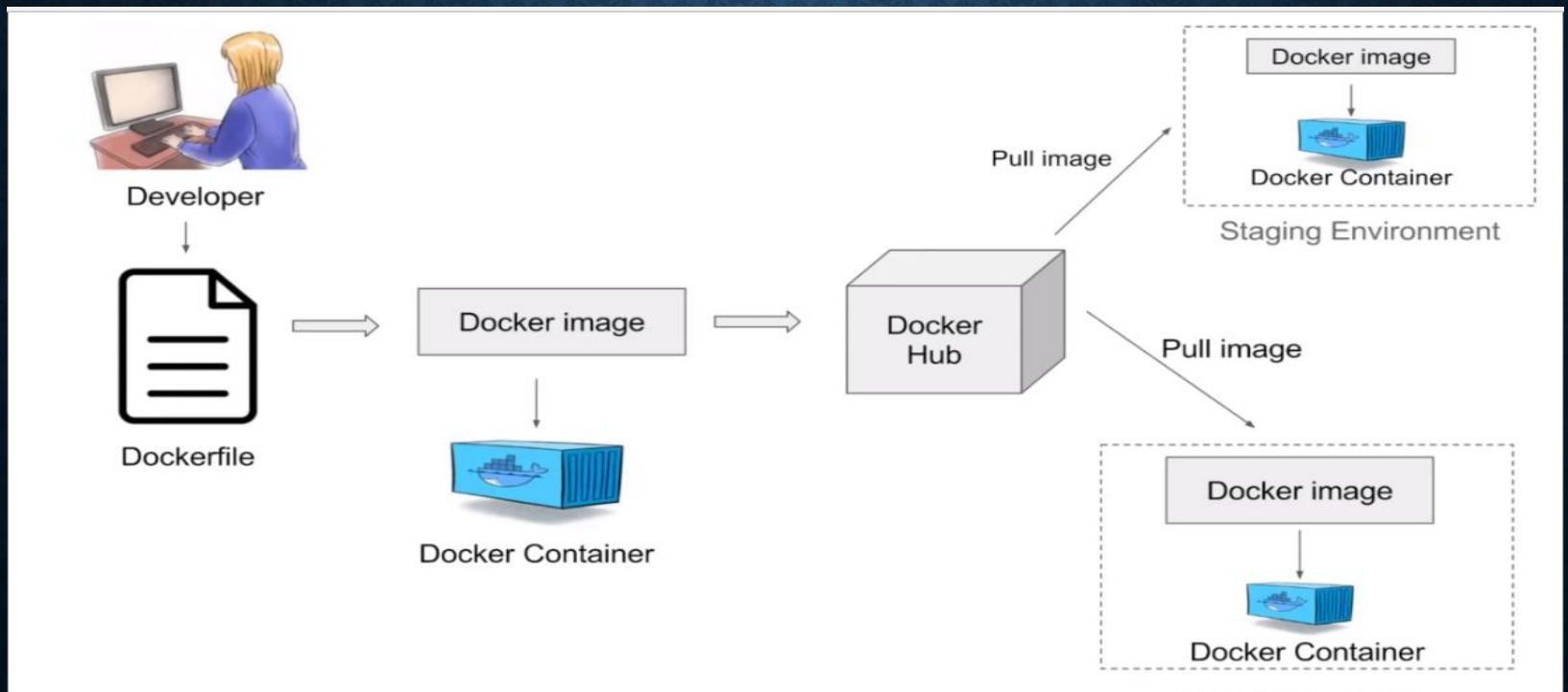
**Docker provides tooling and a platform to manage the lifecycle of your containers:**

* **Develop your application and its supporting components using containers.**
* **The container becomes the unit for distributing and testing your application.**
* **When you're ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.**

# [**What can I use Docker for?**](https://docs.docker.com/get-started/overview/#what-can-i-use-docker-for)

**Docker streamlines the development lifecycle by allowing developers to work in standardized environments using local containers which provide your applications and services. Containers are great for continuous integration and continuous delivery (CI/CD) workflows.**

* **Your developers write code locally and share their work with their colleagues using Docker containers.**
* **They use Docker to push their applications into a test environment and run automated and manual tests.**
* **When developers find bugs, they can fix them in the development environment and redeploy them to the test environment for testing and validation.**
* **When testing is complete, getting the fix to the customer is as simple as pushing the updated image to the production environment.**



**Docker Benefits:**

**• Containerization (OS level virtualization) (No need guest OS)**

**• No pre-allocation of RAM • Can replicate same environment**

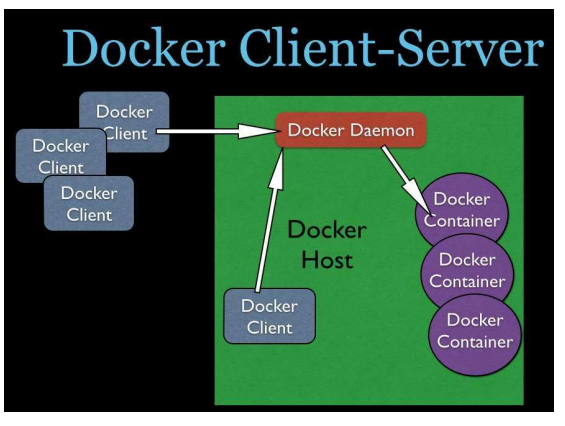
**• Less cost**

* **Less weight (MB’s in size)**
* **Fast to fire up**
* **Can run on physical/virtual/cloud**
* **Can re-use(same image)**
* **Can create machines in less time.**

****

[**Docker architecture**](https://docs.docker.com/get-started/overview/#docker-architecture)**:**

* **Docker is implemented as a client-server system; The Docker daemon runs on the Host and it is accessed via a socket connection from the client.**
* **The client may, but does not have to, be on the same machine as the daemon. The Docker CLI client works the same way as any other client but it is usually connected through a Unix domain socket instead of a TCP socket.**

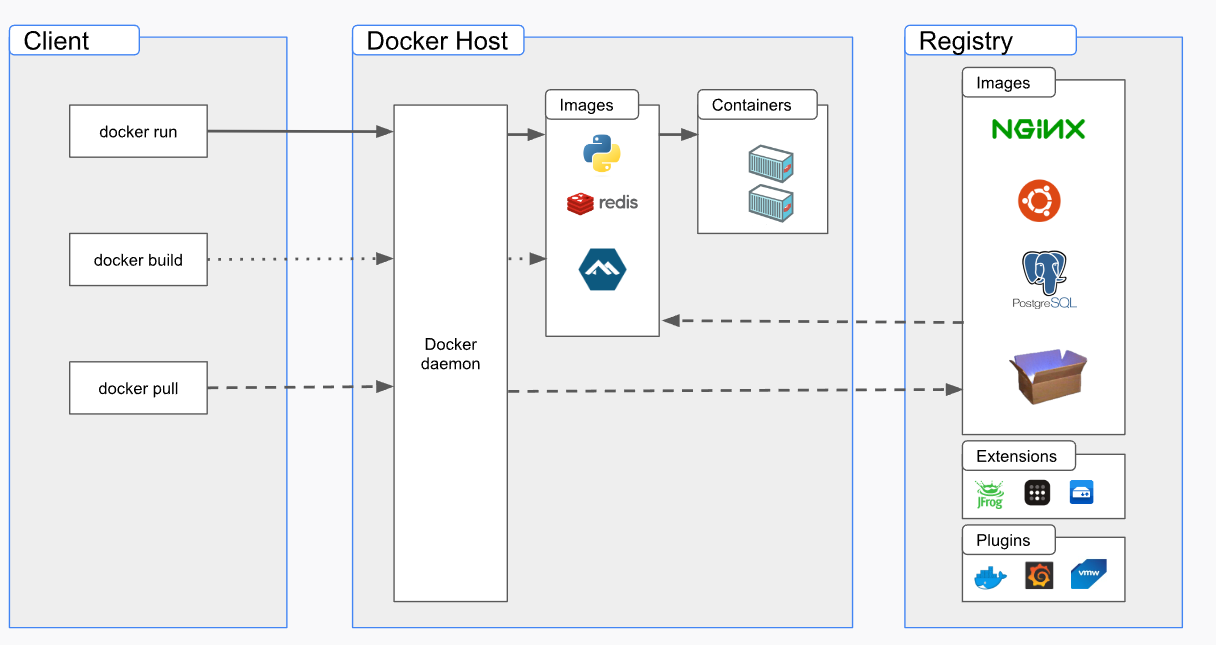
****

**Docker Daemon:**

**The daemon receives commands from the client and manages the containers on the Host where it is running.**

**ϖ Responsible for maintaining Containers & Local Images.**

**ϖ Responsible for pulling the images from Image Registry if required or pushing locally built images to Image Registry.**

****

[**The Docker daemon**](https://docs.docker.com/get-started/overview/#the-docker-daemon)**:**

**The Docker daemon (dockerd) listens for 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.**

[**The Docker client**](https://docs.docker.com/get-started/overview/#the-docker-client)**:**

**The Docker client (docker) is the primary way that many Docker users 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.**

### [**Docker Desktop**](https://docs.docker.com/get-started/overview/#docker-desktop)**:**

**Docker Desktop is an easy-to-install application for your Mac, Windows or Linux environment that enables you to build and share containerized applications and microservices. Docker Desktop includes the Docker daemon (dockerd), the Docker client (docker), Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper. For more information, see**[**Docker Desktop**](https://docs.docker.com/desktop/)**.**

### [**Docker objects**](https://docs.docker.com/get-started/overview/#docker-objects)**:**

**When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.**

**ϖ Host, the machine that is running the containers**

**ϖ**

**1. An image is a file structure, with meta-data for how to run a container**

**2. The image is built on a union file system, a file system built out of layers**

**3. Every command in the Dockerfile creates a new layer in the file system**

**4. When a container is started all images are merged together into what appears to the process as unified. When files are removed in the union file system they are only marked as deleted. The files will still exist in the layer where they were last present.**

**docker run -i -t ubuntu /bin/bash**

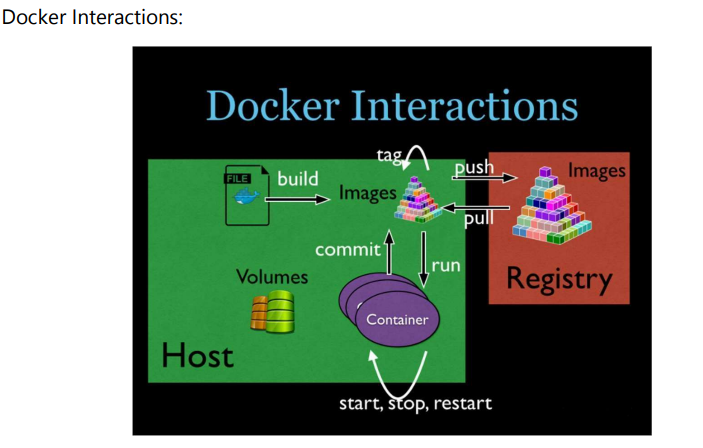
**ϖ Container, a contained running process, started from an image**

1. **When a container is started, the process gets a new writable layer in the union file system where it can execute.**
2. **It is also possible to make this layer read-only, forcing us to use volumes for all file output such as logging, and temp-files.**

ϖ **Registry, a repository of images**

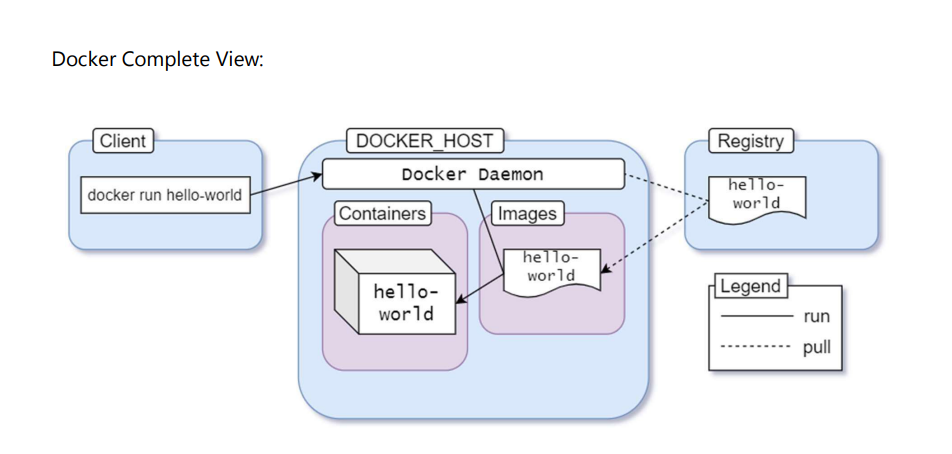
**ϖ Volume, storage outside the container**

**ϖ Dockerfile, a script for creating images**

****

**Installing Docker:**

**1. Go to https://docs.docker.com/engine/install/ and download as per your machine’s Operating System installed [CentOS, Debian, Mac or Windows].**

****

**• Docker Commands :**

**ϖ How to run the application over docker**

1. **Normal Mode:**

**docker run -p 5000:5000 muralisocial123/python-app-sample:1.0**

1. **Detached/Daemon Mode:**

**docker run -p 5000:5000 –d muralisocial123/python-app-sample:1.0**

**ϖ How to check traces/logs:**

**docker run --name test -d busybox sh -c "while true; do $(echo date); sleep 1; done"**

**docker logs containername #entire log.**

**docker logs -f containername #tailing the log.**

**docker logs -f --until=2s test**

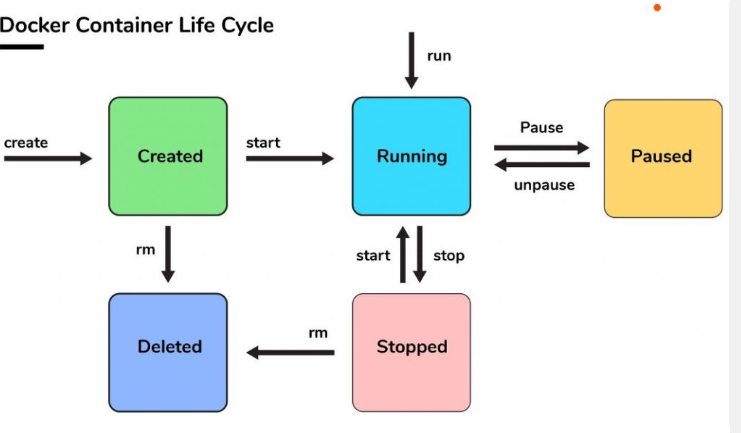
**Container Life Cycle and Commands :**

**1. docker container ls #all the docker images which are up and running**

**2. docker container ls -a #all the docker images which are up and exited**

**3. docker create # creates a container but does not start it**

**4. docker run -p 5000:5000 muralisocial123/python-app-sample:1.0**

**5. docker container stop container\_id #to stop the particular docker**

**container**

**6. docker container start # will start it again**

**7. docker container restart # restarts a container**

**8. docker container pause container\_id #to pause the container – no**

**response from APIs can be observed here**

**9. docker container unpause container\_id #to resume the container –**

**you can see APIs responding back**

**10. docker container inspect container\_id #meta data about container**

**11. docker container prune #this would remove all the stopped**

**containers**

**12. docker container stop container\_id #graceful shutdown of container]**

**13. docker container kill container\_id #container would be terminated as**

**it is**

**14. docker attach # will connect to a running container**

**15. docker wait # blocks until container stops**

**16. docker exec # executes a command in a running container**

**17. docker run -p 5000:5000 -d --restart=always muralisocial123/python-app-sample:1.0**

**#whenever docker daemon restarts, this image would be made available if it was terminated**

**18. So even with restart policy, once stopped & prune command is**

**applied over container, there is no way to bring it up again.**

**Images :**

**1. docker images #shows all images**

**2. docker build #creates image from Dockerfile**

**3. docker image history image\_id #list changes of an image**

**4. docker image inspect image\_id**

**5 . docker image remove image\_id # removes an image from (local)**

**6. docker import #creates an image from a tarball**

**7. docker commit #creates image from a container**

docker tag local-image:tagname new-repo:tagname

docker push new-repo:tagname

**Tags:**

**1. We can create multiple tags for the same image**

**2. docker tag muralisocial123/python-app-sample:1.0**

**3. Latest ……. generally most recent tag & 1.0.0. RELEASE …. latest release**

**Tag**

**Pull:**

**1. docker pull mysql #just pulls the image from Image Registry to our**

**local**

**2. docker run will deploy the image if present in local, otherwise it will**

**pull first and then deploys the image**

** Push:**

** Searching over Image Registry**

**1. docker search mysql**

** Docker Events**

**1. docker events #would help us to know what are all the events**

**happening over docker engine or docker environment**

** Docker top**

**1. docker top containe\_id #lists all the processes running inside that**

**container]**

** Docker stats**

**1. docker stats #memory statistics of all the containers [CPU,**

**MEMORY]**

** Limiting Resources to a docker image [MEMORY & CPU]**

**1. docker run -p 5000:5000 -m 512m –cpu-quota 5000 -d**

**muralisocial123/training/rest-api:1.0.0.RELEASE**

**2. cpu-quota 100000=100% so 5000=5% of CPU for the above command**

** Docker Daemon Management**

**1. docker system df #all the docker daemon management things likes**

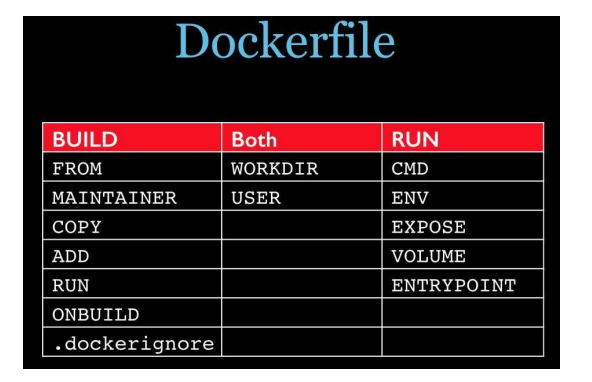
**images, containers, volumes**

**Dockerfile:**

**Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. This page describes the commands you can use in a Dockerfile.**

**ϖ Dockerfile is a configuration file (blue print) which is used to build Docker images from project.**

**ϖ Dockerfile Commands: The Dockerfile supports 13 commands. Some of the commands are used when you build the image and some are used when you run a container from the image.**

****

**BUILD Commands:**

**1. FROM – The image the new image will be based on**

**2. MAINTAINER – Name and email of the maintainer of this image**

**3. COPY – Copy a file or a directory into the image**

**4. ADD – Same as COPY, but handle URL:s and unpack tarballs**

**automatically**

**5. RUN – Run a command inside the container, such as apt-get install**

**6. ONBUILD – Run commands when building an inherited Dockerfile**

**7. .dockerignore – Not a command, but it controls what files are added**

**to the build context. Should include .git and other files not needed**

**when building the image**

**RUN Commands:**

**1. CMD – Default command to run when running the container, can**

**be overridden with command line parameters**

**2. ENV – Set environment variable in the container**

**3. EXPOSE – Expose ports from the container. Must be explicitly**

**exposed by the run command to the Host with -p or -P**

**4. VOLUME – Specify that a directory should be stored outside the**

**union file system. If is not set with docker run -v it will be created**

**in /var/lib/docker/volumes**

**5. ENTRYPOINT – Specify a command that is not overridden by**

**giving a new command with docker run image cmd. It is mostly**

**used to give a default executable and use commands as**

**parameters to it.**

**Both BUILD and RUN Commands:**

**1. USER – Set the user for RUN, CMD and ENTRYPOINT**

**2. WORKDIR – Sets the working directory for RUN, CMD,**

**ENTRYPOINT, ADD and COPY.**

**Docker Project :**

**Deploy a Web Application using a Dockerfile.**

**Step1 : Create a directory.**

**Step2: Create a Web application Page and Deploy into Nginx Server.**

**Step3: Build a Image and Deploy in a Container**

**Task1 :** [**https://docs.docker.com/get-started/07\_multi\_container/**](https://docs.docker.com/get-started/07_multi_container/) **: Project**

**Task2:** [**https://docs.docker.com/get-started/09\_image\_best/**](https://docs.docker.com/get-started/09_image_best/)

**docker run --rm Ubuntu: trusty ping google.com**

**FROM Ubuntu:trusty**

**CMD ping –c 4 google.com**