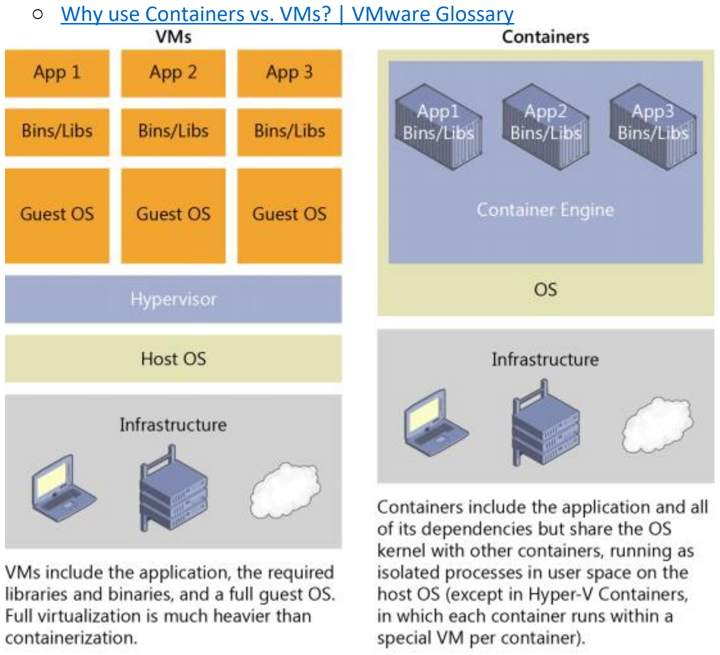
Docker-Basics

- **Containers**
  - **Introduction:**
    - **Definition:** Containers are executable units of software in which application code is packaged, along with its libraries and dependencies, in common ways so that it can be run anywhere, whether it be on desktop, traditional IT, or the cloud.
    - In other words: Lightwight & portal implementations of applications running in their own app / OS space.
  - **Container Vs Virtual machine:**
    - Why use Containers vs. VMs? | VMware Glossary


VMs
Containers
App 1
App 2
App 3
Bins/Libs
Guest OS
Hypervisor
Host OS
Infrastructure
App1 Bins/Libs
App2 Bins/Libs
App3 Bins/Libs
Container Engine
OS
Infrastructure
VMs include the application, the required libraries and binaries, and a full guest OS. Full virtualization is much heavier than containerization.
Containers include the application and all of its dependencies but share the OS kernel with other containers, running as isolated processes in user space on the host OS (except in Hyper-V Containers, in which each container runs within a special VM per container).

Img src:What is Docker? | Microsoft Docs

- In summary

Virtual Machine	Container
Virtualize Physical Hardware	Virtualize Underlying OS
Requires Hypervisor to run	Requires Container Engine to run
Has individual Kernel	Shares Kernel with Host OS (i.e the host provides the kernel)
Heavy & High load time	Lightweight
Can run different OS	Running single OS (there are alternates possible that allow Linux and Windows containers to work together)

  - There are many tools now in market that work with containers. Docker Alternatives for Containerization & Their Standout Features (simplilearn.com)
    - Example rkt coreOs (Rocket),MESOS (Apache) , Docker etc.
  - Trivia:
    - Linux Vs Windows containers:
      - Traditionally Containers meant Linux Containers up until 2016.
      - From about 2016 MS started adding more support for containers, hence you now see Linux Containers & Windows Containers.
      - Concepts are the same, but the binaries might be slightly different.

- **Introduction to Docker:**
  - Opensource container management software, written in "Go".
  - Documentation : https://docs.docker.com
  - Further read - https://github.com/veggiemonk/awesome-docker#hosting-images-registries
  - Started by a company called dotCloud as an opensource project. Company would later become Docker Inc.
  - Docker tool used to manage and work with Containers.
  - Two editions: http://docker.com/pricing & Announcing Docker Enterprise Edition - Docker Blog
    - Docker Community Edition (Free)
    - Docker Enterprise Edition (Paid)- This is now bought over by Mirantis & the docker engine integrations with Swarm will be replaced at some point with Kubernetes. What We Announced Today and Why it Matters | Mirantis
  - Release
    - Edge (beta release) - Usually in a monthly cycle.
    - Stable (prod release) - Support / rollout is usually 3 rolling periods of edge cycles, with about a month or so overlap for upgrade.
  - Three Types:
    - Tools to Older Mac & Windows:
      - Docker Machine (sometimes loosely also called docker toolbox):
        - Different install build, works slightly different. Docker creates a VirtualBox VM and then runs container in it, with NAT to internet)
        - Docker Machine overview | Docker Documentation
        - Machine was the only way to run Docker on Mac or Windows previous to Docker v1.12. Starting with Docker v1.12, Docker Desktop for Mac and Docker Desktop for Windows are available as native apps

            - Used for Windows / Mac where Virtualization is not supported.
            - Windows 7,8,9, 10 Home
            - Mac less than OSX Yosemite 10.10.3
            - Does not support Windows Containers.
        - Docker Toolbox (Installer for old versions)- depreciated
            - Further read - Docker Toolbox | Docker Documentation
    - Direct Installs
        - Docker Desktop for Windows: Docker Desktop for Windows user manual | Docker Documentation
            - Installable for Windows 10 Pro & Enterprise , Windows Server 2016. Supports Windows Containers.
        - Linux local install (traditionally it was only Linux, until Windows 2016).
            - Recommended to install from docker store, instead of within Linux Distro (as it might be old).
            - Docker's automated script to add their repository and install all dependencies
                - curl -sSL https://get.docker.com/ | sh
        - Docker Desktop for Mac: Docker Desktop for Mac user manual | Docker Documentation
            - Mac great than OSX Yosemite 10.10.3
    - Cloud - docker build specific to the cloud (like AWS, Azure and GCP).
- Lab & Other references:
    - Official Docker Repo: https://github.com/docker
    - Tutorials & Labs:
        - https://github.com/docker/labs
        - https://training.play-with-docker.com/
        - Play with Docker (play-with-docker.com)
        - https://dockerlabs.collabnix.com/

- **Docker Vs Docker Compose Vs Docker Swarm vs Kubernetes**
    - Further Read: https://www.techrepublic.com/article/simplifying-the-mystery-when-to-use-docker-docker-compose-and-kubernetes/ https://docs.docker.com/engine/swarm/ & https://www.bmc.com/blogs/kubernetes-vs-docker-swarm/
    - Definition:
        - Docker is (in many cases) the core technology used for containers and can deploy single, containerized applications
        - Docker Compose is used for configuring and starting multiple Docker containers on the same host--so you don't have to start each container separately
        - Docker swarm is a container orchestration tool that allows you to run and connect containers on multiple hosts.
            - Not to be confused with Swarm Classic (which is not in prod any more). This is now referenced in Docker as "Swarmkit" https://github.com/docker/swarmkit
        - Kubernetes is a container orchestration tool that is similar to Docker swarm, but has a wider appeal due to its ease of automation and ability to handle higher demand
    - When to use:
        - Docker - when you want to deploy a single (network accessible) container
        - Docker Compose - when you want to deploy multiple containers to a single host from within a single YAML file
        - Docker swarm - when you want to deploy a cluster of docker nodes (multiple hosts) for a simple, scalable application
        - Kubernetes - when you need to manage a large deployment of scalable, automated containers
- **Docker Machine:**
    - Machine was the only way to run Docker on Mac or Windows previous to Docker v1.12.
    - Starting with Docker v1.12, Docker Desktop for Mac and Docker Desktop for Windows are available as native apps
- **Components of Docker**

Docker Hub	• Public image repository. • Requires authentication
Docker Engine	• Central Application that helps run the containers (like a Hypervisor for VMs) • Manages container run times. • Shares Host OS resources
Docker Image	• Template that contains the package with details of OS, application etc. • Any changes or customizations to image, creates a new version • Hosted in Docker Repo (can be public like Docker Hub or local Repo) • Multiple containers can be run using a single image • Reference examples • Mysql [latest image] • Mysql:2 [image with a particular version, if no version latest is assumed] • Myname/mysql [image uploaded by a particular user]
Container	• The Virtualized running application. • Created based on a docker image. Can have many containers based on a single image • Configuration of container is in docker file
Docker Volume	• Storage for containers to ensure data availability post container destroyed • Can be shared across containers. • Default volume created in container (does not have data persistency)
Dockerfile	• Configuration template file used to define how a container needs to behave: ports to open, image to be used, commands to be executed etc. • Created in YAML. • Used to run customer containers.

- **Terminology:**
    - Attach/ detach a container - Associate / disassociate the container from the Standard Input / Out put / error terminals (STDIN/STDOUT/STDERR)

- **Docker Commands**
  - Typical Format : Docker <Command> <sub-command> [parameters] (new way)
    - The set of commands are called "Management Commands" followed by the options
  - Before 2017 the format was : docker <command> [parameters] (old way)
  - Both formats work.
  - --help or -h will provide help on the respective command / sub-commands
  - Cheat Sheet: docker cheat sheet & The Ultimate Docker Cheat Sheet | dockerlabs (collabnix.com)
- **Client Commands:**

$ docker version	• Displays the docker version (client & server). • Client = Docker client • Server = Docker Engine
$ docker info	• Displays server engine configuration and runtime status like (containers, swarm, volume etc)
$ docker system	• Used to manage docker. • Further Read: https://docs.docker.com/engine/reference/commandline/system/

- **Docker container:**

$ docker run	• Runs an image (as container) • Always starts a new container • Usage: $ docker run <docker_image_name> [options] • Variations: • docker run -name <container name> -p <local port>: <container port> <image name> [starts a specified container with a name, instead of just IDs and maps local to container ports] • e.g : docker container run –name web -p 5000:80 alpine:3.9 [Run a container from the Alpine version 3.9 image, name the running container “web” and expose port 5000 externally, mapped to port 80 inside the container.] • Parameters (optional) : • --detach - run in background (if not specified, logs will be shown on console) • --name [name] (name of container) • -- env ENV_NAME=VALUE (pass environment value to container) • -- publish <Host Port> :<container port> • -I (Interactive)- leaves STDIN available • -t (tty term) - Allocates a psuedo tty term (usually -it is provide to have an interactive terminal session with a container) ○ This runs the default command in interactive mode. ○ Can be specified alternate command, In those scenarios when the command ends the container stops. • --rm : Container is removed after the container exits. • --net <network name> - Adds container to specified network • --net-alias search -> Adds the container in a DNS round robin access when the specified alias is accessed.. • New format: • $ docker container run <docker image> [options] • $ docker container run -- publish 80:80 --detach nginx [runs latest nginx image from docker hub in background (--detach), and displays container's website on local hosts port 80 mapping.
$ docker ps	• Lists the containers running • Variations: • $ docker ps -a [lists all containers, including stopped] • $ docker ps -l [lists last created container] • New format (same result) • $ docker container ls
$ docker start	• Starts an existing container that has been paused / stopped.
$ docker stop	• Stops a container from running (does not remove the container) • Usage: $ docker stop <container_id> • New format: • $ docker container stop (both work)
$docker rm	• Removes a container • Usage: $ docker rm <container_id> • Needs to be stopped before it can be removed. • Alternately the switch of "-f" can be used to force remove • New format: • $docker container rm
$ docker exec	• Used to run commands inside a container • Usage: $ docker exec [switch] [container_ID] [command] • For e.g. to login to container • $ docker exec -ti ae45cd bash [basically saying to run bash shell in interactive mode on the container] • Note: Unlike docker run -it , ending the command does not stop the container as Exec runs a process on top of the current command set , so the default command set that kicked off the start of the container is still running

$ docker top	• Lists process in a container (takes container Name as parameter) • Usage: • $ docker container top (or docker top) <container name/ Id>
$ docker logs	• Display logs of the container • Usage: • $ docker logs <container name> or • $ docker container logs <container name>
$ docker inspect	○ Lists config of container , meta data associated with container in JSON format. ○ $ docker container inspect <container name/ Id> ○ Or $ docker inspect <container name / ID >
$ docker stats	• Provides live streaming view of performance data of containers (or if a container ID/Name is provided, it will provide for that specific container- CPU, Memory, Network, I/O)
$ docker update	Updates the resource utilization for a container (CPU / Memory etc)

- **Docker Networking:**
  - **Concepts:**
    - A Virtual IP is automatically assigned by Docker to the container.
    - If no port mapping between container & host is provided, no ports will be opened.
    - There is a default virtual network within the docker engine called "Bridge" or sometimes called "docker 0".
      - Unless otherwise mentioned, all containers created will be attached to this default virtual network.
      - This has the NAT'd to the host network
    - There is another type call the "Host" network, which basically is not network but a representation that the container is directly attached to the host IP . Not secure, but has performance advantage.
    - Containers on same virtual network can talk to each other , without having to have their ports opened. (Think of port publish as opening the box to external traffic)
    - If you would like two containers not to talk to the rest of the containers, you create another Virtual network and have these containers use them , that way they will be isolated from the rest of the containers in docker.
  - **DNS:**
    - Docker demon has a default DNS server that the containers use.
      - Note: Docker defaults the hostname to the container name , but can be changed through aliases.

  - **Commands**:
    -

$ docker port	• Shows how port mapping between container -> host is setup. • Usage : $docker container port <container ID/name>
$ docker network ls	• Lists all virtual networks. • Usage: • $ docker network ls
$docker network inspect	• Inspects/ lists config of a particular Network • Usage: $ docker network inspect <network name>
$docker network create	• Creates a network. Has optional switch to specify network driver
$ docker network connect	• Attaches a container to network. When doing so creates a NIC with IP properties of the network to be attached (CIDR etc). Similar to adding a NIC while the box is running. • Can have a container attached to multiple networks.
$ docker network disconnect	• Disconnects a container from a network
$ docker network rm	• Removes one or more networks

- **Docker Images**
  - **Concepts:**
    - Image= Application Binaries & Dependencies + Metadata of the binaries & dependencies + How to run it
    - A registry is a central place where the images are made available. It can be either a public registry (open to all) or a private registry (or restricted use, like only within an organization).
      - Docker Provides 3 tools - Docker Hub, Docker Enterprise DTR & Docker Registry (Private)
      - There are other 3rd party registries available - https://github.com/veggiemonk/awesome-docker#registry
    - An image in a registry file is called a repository (like the repo in Github or other versioning system) as there will be multiple versions of the same image.
      - Public Image: Anyone who has access to the registry can access this image.
      - Private Image: Personal version of the image, not accessible to all. The repo needs to be created first before any private i mage can be uploaded to this repo.
    - An Image in any registry tool(like docker hub) is identified by Name, Tag, ID
      - Name: Repository Name is the name that the image is identified with in the registry (Note: Not using the term Image Name here , as Image Name is the customized name given to the image, when it is pulled from DockerHub).
      - ID - Is a unique ID given to an image
      - Tag: Tag is a reference for the image (a label) usually used to represent the Version. if no tag is mentioned, latest is assu med.
        - Image can have multiple tags.
    - Official images have the format of <repository name>:<tag name>
    - Unofficial images have format of <account name>/<repository name>:<tag name>.
      - Account name can be an user name or an org name.
    - Private registries (by default the DockerHub is the de-facto registry , but if you would like to use a different registry (for push or pull of images) like a corporate registry or else ware. The images need to be tagged as "<Hostname>/<account name> / <image name>: <tag

name>". This will fetch or push images from the registry mentioned in "Host Name". If the host is listening on a different port, that needs to be mentioned as well
    - Example: Myhost.abc.com:5000/myaccount/myimage:1.0
  - Note: Registry needs to be secured (like a TLS Certificate etc).
  -
  - There is a container called Registry that can be used to enable local private registry. If this is used, care needs to be tak en about security and also to map a volume path to store the image and meta data.

- **Commands:**

$ docker images	• Lists all the images on local host
$ docker build	• Builds image from a docker file • Usage : docker build -t <image name> [-f <docker file path]
$ docker commit	• Builds image from a container file • Usage: docker commit [CONTAINER_ID] [new_image_name]
$ docker save	• Exports image • Usage: $ docker save -o <complete_tar_file_path> <docker_image_name>
$ docker rmi	• Removes docker image • Usage: $ docker rmi <docker_image_id>
$ docker login	• Logins to specified registry server . If no server is specified logs into Dockerhub • Usage: • $ docker login <server URL> (Server URL is provided when we want to use a different docker registry , like a company registry, instead of DockerHub) • Creates an authentication key stored in .docker/config.json . • Hence it is important to logout, especially if you are on shared machines.
$ docker logout	• Logout from the logged in registry server
$ docker pull	• Pulls image from registry (can be private or public like Docker Hub) • The pull is based on image ID. If the image ID is not available in local cache only then it pulls the image, else it just adds the entry to the image lists. • For e.g; • $ docker pull nginx:latest -> pulls the latest image with image ID as say abcde. ○ Assume that this image has another tag with say 1.1 ○ Running $ docker pull nginx:1.1 will not get another copy. Instead it will say this image is already available. And when you run ○ $ docker image ls (to list the images) ▪ You will see two entries for nginx:latest and nginx:1.1 both having same image ID.
$ docker push	• Pushes image to registry(can be private or public like Docker Hub)
$ docker history	• Shows the changes made to a particular version of an image. • Usage & example : • $ docker history nginx:latest (provides details all the changes made to the base image, across all versions / revisions)
$ docker image inspect	• Provides meta data of an image • Usage : • $ docker image inspect nginx:latest

$ docker tag
or
$ docker image tag
	Assigns a tag to an existing image.

$ docker image prune	Cleans up unused images Usage: $ docker image prune -a (cleans up all unused images)

- **Dockerfile contents**

  Concepts:
  - Customizes the image that we would like to deploy.
    - If not specified, Docker will deploy the un-customized image available in DockerHub or other image repository.
  - The Dockerfile contains the instructions for customizations (or new image definitions) in a paragraph format (unique to Dockerfile).
    - Default file is Dockerfile, but can have any file name with that format (in which case when building "-f <filename>" switch options are to be used)
    - Each paragraph represents a layer in image, so sequence of execution needs to be careful as Docker runs it top - down.
    - If the Dockerfile has changed and we rebuild the image. Docker will not re-execute steps that havent changed. However When a line in Dockerfile changes, Docker rebuilds from that line down to EOF (even if there is no change in another steps)
  - Best Practise:
    - Best to keep the items that change the least at the top and change the most at the bottom.
- Further read: (https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#dockerfile-instructions & https://docs.docker.com/engine/reference/builder/
- **Commands in Dockerfile:**

FROM	Image to base the container on
RUN	Commands to run (usually used to install additional packages on top of the image) • Multiple commands can be run in a single line.

	• RUN <command 1> ; <command 2> • Or • RUN <command 1> && <command 2> • The difference is "&&" runs command 2 only after command 1 is successful and ";" runs command 2 regardless of success of command 1
CMD	Command to run if no parameters are passed at docker run Runs at right after container initialization
WORKDIR	Sets the working directory within the container. Preferred method instead of having to use " RUN cd <directory path>\<directory name>"
ENTRYPOINT	Command to run right after container initialization. Is not impacted by the parameters from docker run command
EXPOSE	Ports on container that need to be open for communication. Will be used later for mapping to the host ports during docker run
ADD	Adds file / path from local host to container. Allows for Http Urls as well.
COPY	Adds file / path from local host to container Format: COPY <Host Directory / file> <Container Director / file> • E.g. Copy . . -> States copy all files in current directory of the Host into the current directory of the container
ENV	Sets environment variable in the container
VOLUME	Instructs docker to create a volume and map to a path in the container. format : VOLUME <Path in container >

Run vs EntryPoint vs CMD : While all 3 are used to execute commands there are executional differences between the three.

- RUN -> Is used to execute commands before the initilization of container. Typically for application installs. Run creates a new layer on top of the image and the subsequent commands are run on it.
- CMD - Command to run if no parameters are passed as part of docker run.
- EntryPoint -- Command to run , does not matter if parameter is passed during docker run or not.

Dockerfile examples:

- Further read: https://docs.docker.com/samples/ & https://github.com/dockersamples
- Example-1
```
FROM nginx
ENV AUTHOR=Docker
WORKDIR /usr/share/nginx/html
COPY Hello_docker.html /usr/share/nginx/html
CMD cd /usr/share/nginx/html && sed -e s/Docker/"$AUTHOR"/ Hello_docker.html > index.html ; nginx -g 'daemon off;'
```
- Example-2
```
# escape=`
FROM microsoft/iis:nanoserver-sac2016
SHELL ["powershell", "-Command", "$ErrorActionPreference = 'Stop'; $ProgressPreference = 'SilentlyContinue';"]

COPY ./share/StaticHtml.zip .

RUN Expand-Archive -Path StaticHtml.zip -DestinationPath C:\StaticHtmlApp; `
   Remove-Item StaticHtml.zip

RUN Import-Module IISAdministration; `
   Remove-IISSite -Name 'Default Web Site' -Confirm:$false; `
   New-IISSite -Name web-app -BindingInformation "*:80:" -PhysicalPath C:\StaticHtmlApp
```

- **Storage**:
  - Further Read - https://docs.docker.com/storage/
  - Concepts:
    - Term
      - Immutable Infrastructure - We don't change the infrastructure when it is running.
      - UFS - Union File System - File system within a container.
    - Since containers are dropped and recreated. There is a need to store application or user data that needs to be accessed even after the container was destroyed or recreated (so that there is no loss of data).
    - There are two options
      - Volumes : Storage outside of the container
      - Bind Mounts - Mounting / linking external storage to the container. Basically linking container path to host path.
    - **Volumes**:
      - Storage outside of container.
      - Deletion of container , delete the volume. Needs to be manually deleted.
      - Referenced in Dockerfile using the VOLUME command. (or using the -v <path> in docker run)
        - E.g. VOLUME /var/lib/mysql
      - By default volumes don't have any names, so it becomes difficult to know which volume is used by which container (especially in a situaion after the containers were removed).
        - So to come out of it you can assign names to volumes using <name>: <local container path> , either in the VOLUME command in Dockerfile or when assigning it run-time using "-v" in "docker run"
        - Would also try to give a meaning full path with name for the volume.
      - Commands

$ docker volume prune	Cleans up unused volumes
$ docker volume ls	Lists the volumes

$ docker volume inspect	Provides details about the volume. Takes the volume ID / volume name as parameter
$ docker volume create	Creating a volume outside of docker run or Dockerfile. Used to create volumes with a specific driver or some other options (you cannot provide either of them during docker run or using VOLUME in Dockerfile

- Bind Mounting:
  - Mapping of existing file or director on Host to a file or directory in a container. It is like two locations pointing to a same file.
  - Since they are host specific, cannot be provided in Dockerfile. Will have to be provided as parameters at run time.
    - Docker run -v <host path>:<container path> (same as -v that we use for volume mapping except that this time it is mapping folder paths and starts with a "/" (linux) or "//" (windows))
  - Full path is always required on both sides.
  - Basically creates a link between the two directories / files and can see modifications .
  - Deleting the container does not delete the contents, it just severs the link. Files in that directory are still there.

- **Docker Compose**:
  - Further Read: https://github.com/docker/compose/releases
  - Concepts
    - A tool used to help run multiple containers, setup their network, storage , firewall , binding dependencies.
      - Kinda like a tool that helps spin multiple containers for an application suite (like one for SQL, one for Web, one for messag ing etc and then put down how they all talk to each other, share information etc).
      - Helps in situations where instead of running multiple "docker runs" with various configs, the configs are stored in one file and executed together.
    - Has two components
      - A Docker Compose YAML file . Default file is docker-compose.yml (can be changed when triggering docker-compose)
      - A Docker Compose CLI to run. Command is 'docker-compose'
    - Used more for Dev / Test environments. As of 1.13 and above, the YAML can be used directly by Docker & consumed by Docker Swarm.
    - A separate binary from docker. Comes bundled with Docker for Windows/Mac & Toolbox but for linux needs to be downloaded separately.
    - Containers are referred to as Services in the compose file.
    - Service Names are more like labels for the containers.
    - Docker Compose YML Format (Further read - https://docs.docker.com/compose/compose-file/) :
      Notes: Version - v2.x is actually better for local docker-compose use, and v3.x is better for use in server clusters (Swarm and Kubernetes)

```
Version : 3  # version of docker compose to use (if not provided, will default to 1)
Services  :
  <service name>:  # referencing a name that will represent a container (will be used by container as DNS name to communicate)
     build:
        context: <directory where the dockerfile is in and build working directory, "." signifies dockerfile is in the same dir as compose file >
        dockerfile: <name>
     image : <image name>: <tag>
     ports:
        - "<host port>:<container port> " example - "8080:80"
     volume  :
        - <host path> : <container path>:[options] (if trying bind mount)
        - <host name>: <container path>:[options] (if trying named volume)
      environment:
        - <env variable >: <value>
Volumes:                # declaration section of volumes , required if volumes are referenced/ used in the services.
   - <volume name>:
etc
```

  - Docker Compose will check for image in local cache and if it is not available then it will download from registry.
    - However, if there is a build block, then instead of pulling from registry, it will build a new image based on the configuration provided in the Dockerfile.
  - Commands:

$ docker-compose Up	Starts containers , volumes, network etc per definition of the compose yml file
$ docker-compose down	Stops containers, cleans up network etc.
$ docker-compose build	Build images

- **Docker Swarm**:
  - Further Read - https://docs.docker.com/engine/swarm/services/ & https://dockerswarm.rocks/
  - **Concepts**:
    - Cluster Management system. Different from earlier Swarm Classic (deprecated)
    - Integrated into the Docker binaries.
    - Needs to be initialized / enabled for it to be used .
      - Can check in 'docker info " output for swarm enable status.
    - Utilizes TLS certificates and encryption for transit.
    - Has Manager - Workers node architecture. Nodes can be promoted / demoted, added - removed etc based on requirement.
    - You can have multiple manager nodes, but only one leader at a time. All manager nodes, share the same information in a database called Raft database. Further read - https://docs.docker.com/engine/swarm/admin_guide/
      - Raft algorithm is used by Kubernetes as well.
      - Raft is primarily used for Quorom(leader), log management, security
    - Note: Worker nodes cannot perform Swarm management commands.
    - **A Service is created (has details of how many replicas we want, definitions of the containers etc) -> That creates a task -> task creates a container per spec.**

- Networking:
  - Overlay Network - Multi Host - Container to container network in a swarm, with optional encryption. *(docker network create --driver overlay).* This is another type of "bridge" network that enables container to container communication , except this helps with communication across different nodes / servers / hosts.
    - Routing Mesh : An ingress network that allows you to access the service in a swarm even if that service is not running on that node.
      - Further Read - https://docs.docker.com/engine/swarm/ingress/
      - Container to container traffic management (traffic routing / load balancing). Internal management where the nodes will route internally to the right container for accessing the service.
      - Built in / out of the box feature. No need to enable.
      - Limitations:
        - Operates at Level 3 (TCP), so cannot have two sites listening to same ports for the swarm
        - Stateless load balancing (so cannot support cookies or affinity).
      - Limitations can be overcome by putting a Nginx or other LB Proxy in front of the swarm so that it perform the required LB activity and then routes the traffic to this routing mesh in Swarm.
  - Note: Swarm puts a internal VIP before the services and the services are accessed through that VIP.
  - With a docker service, persistant data storage you would have to use the --mount option with source and target to achieve the same.
    - Usage: -- mount type=volume source=<name of volume> target=<path of directory for persistent data storage> (and if using // for the source it would be bind mount and the type will be changed accordingly)
- **Best Practices :**
  - It is not recommended to build the images in a swarm (especially in production), it is best to have images built either in th e CI/CD steps or have it already built and pushed in registry.

- Commands

$ docker swarm init	Initializes docker on current installation to start working in swarm mode Starts current node as master node (with leader) Issues token that can be used to have a node join the swarm cluster.
$ docker node	Manage nodes (promote / demote, add/ remove, list)
$ docker swarm	Manage swarm cluster (initialize, join / leave, manage tokens)
$ docker service	Similar to a docker run but more of a multi container level Add remove services, list tasks , service information etc.
$ docker service create	Creates a service based on an image name If no service name is added, a random name is generated
$ docker service update	Updates an existing service with new parameters (like number of replicas etc) Takes Service name / id as parameter Usage : $ docker service update <service name> [options]
$ docker service ls	Lists the available services
$ docker service ps <service name>	Lists the tasks running for that service across all nodes
$ docker node ps [node name]	Lists the tasks running on current node (if no node name specified) Else will provide the tasks running on a specified node.
$ docker node update --role	Updates the current role of the node (between worker & manager)
$ docker service join-token	Gives the token required to join the network for a worker or a master. Tokens can be rotated as well in case the token is compromised.

- **Docker Swarm Stack (or just stacks):**
  - **Concept:**
    - **Swarm Stacks:** It is the compose files implementation in Swarm. Swarm can accept compose files for Services, networks and volumes for implementation.
    - Cannot use build section in Compose File file (if this file is to be used in Stack) , use deploy section instead. Build secti ons are ignored during stack deployment. Docker-compose ignores the build sections. (so the same file can be used for both in docker-compose and docker-stack)
      - Compose Template (the compose file) version has to be atleast 3
    - There is no update, so if the compose file changes, rerun the stack deploy again. If the stack already exists, swarm will aut omatically recognize it and make the necessary changes
  - Commands:

$ docker stack deploy	Creation of services based on compose file Usage: $ docker stack deploy -c <docker compose file> <stack name>
$ docker stack ls	Lists the stacks
$ docker stack ps <stack name>	Lists services in a stack

- **Secrets**:
  - Concepts:
    - Swarm Raft DB is encrypted by default and the connection between manager & worker is a TLS + PKI auth secure channel. Swarm secrets leverage this.
    - Secrets are stored in Swarm and assigned to Services. Secrets are visible to only the containers of the services mapped.
    - Enabled out of the box, no special config required.
    - Apps see the secrets as a file on disc but they are actually files in memory.
      - Path is /run/secrets/<secret name> (where secret name is the file name & content of the file is the encrypted information).

            - If this is to be mapped to a key : value store, key is the secret file name & value is the content of the file.
        - Associated with service using the "--secret <secret name>" switch when creating the service using "docker service create", so that the container / service can work with them (remember they will be seeing the secrets as files, so will have to be processed a s such).
            - For e.g. you could pass the password as an environment variable something like
                - -e PASSWORD_FILE=/run/secrets/<secretName> (but that should be in the image definition so need to check before using)
        - Secrets are part of the immutable infra in services, so if secrets change or are removed from service, service will terminate the container and redeploy the container.
        - To use Docker Swarm Stacks with Secrets, the compose template should be >= 3.1
    - Command:

$ docker secret create	Encrypts a file or text and stores in database with a secret name that will be referenced by other services or containers. Usage: $ docker secret create <secret name> <file to encrypt>
$ docker secret ls	Lists the secrets
$ docker secret inspect <secret name>	Provides meta data of the secret (ofcoz the encrpyted content is not shown)

- Docker Health checks
    - Concepts :
        - Built in features that is supported by Compose, Docker Run, Swarm etc.
        - Not a replacement for the monitoring tools, but does the preliminary checks.
        - Runs a commands inside the shell, so even if ports are not exposed, it will still work.
        - 3 States: Starting, healthy & unhealthy.
        - Actions can be performed based on health check status, however this can be done only in the Swarm / Stack mode.
    - Commands:
        - Docker run : Uses the options during docker run
            - Usage :
                - Docker run [docker switches] <image> [Health Check Options]

--health-cmd=<command>	- command to run in container for health check like a curl access to the website from within container etc.
--health-interval=5s	- duration to run the command (in seconds; like "5 s" etc
--health-retries=2	- no of retries before flagging as unhealthy
--health-timeout=2s	- gap between attempts in seconds
--health-start-period=15s	- gap between starting the health check and container initialization

        - Dockerfile: HealthCheck command in Dockerfile
            - Usage (single line, a "\" is used to signify continuous of the current line in next line)
                - HEALTHCHECK [options] \
                  CMD <Command to run>
            - Options:
                - --interval=<duration>
                - --timeout
                - --retries
                - --start-period
        - In Compose File / Stacks (needs atleast compose template version of 2.1 for using health checks):
            - Example Usage:

```
web:
image: nginx
healthcheck:
  test: ["cmd","curl", "-f", "http://localhost"]
  interval: 1m30s
  timeout: 20s
  retries: 3
  start_period: 15ss
```