**1. What is Docker?**

Docker is an open source containerization platform designed to make it easier to create, deploy and run applications using containers. to ensure that your application works seamlessly in any environment be it development or production.

**2. What are Docker Images? How to list all images?**

Docker images are special build images which contains all the necessary files library and packages. They are read only images used to make containers. Docker images does not have a state and never changes.

Run below command to list all images on host.

$ docker image ls

**3. What are Docker container?**

Docker containers are run time instance of an image, it contains everything needed to run an application.

Run below command to list all containers on host.

$ docker container ls –all

**4. How is a Docker container different from a hypervisor?**

The hypervisor is a software allowing the abstraction from the hardware. Every piece of hardware required for running software has to be emulated by the hypervisor Because there is an emulation of the complete hardware of a computer. Hypervisors may set up on the hardware directly or on a host operating system.

Container-based virtualization utilizes kernel features to create an isolated environment for processes. In contrast to hypervisor-based virtualization, containers do not get their own virtualized hardware but use the hardware of the host system. Therefore, software running in containers does directly communicate with the host kernel and has to be able to run on the operating system and CPU architecture the host is running on.

**5. Can we run multiple apps on one server with Docker?**

Yes we can run multiple apps on one server with docker, Docker works well with microservices, we can run multiple containers from docker images on one server. But The size of your applications as well as the amount of available resources (cpu, memory etc..) will all affect the number of containers that can be run in an environment.

**6. What is Docker Hub?**

The Docker Hub is a centralized resource for working with Docker and its components. It provides the following services:

* Docker image hosting
* User authentication
* Integration with GitHub and Bitbucket

**7. How to login Docker Hub? Command to push your image in registry/DockerHub?**

Prerequisite: Sign up t docker hub. (https://hub.docker.com)

$ docker login <docker-registry>

$ docker push [option] <image-name>:[tag]

**8. What are the main features of Docker Hub?**

Image Repositories: Find and pull images from community and official libraries, and manage, push to, and pull from private image libraries to which you have access.

* Image Repositories: Find and pull images from community and official libraries, and manage, push to, and pull from private image libraries to which you have access.
* Automated Builds: Automatically create new images when you make changes to a source code repository.
* Webhooks: A feature of Automated Builds, Webhooks let you trigger actions after a successful push to a repository.
* Organizations: Create work groups to manage access to image repositories.
* GitHub and Bitbucket Integration: Add the Hub and your Docker Images to your current workflows.

**9. How to check Docker version?**

$ docker –version or

$ docker version

**10. How to get detailed information about docker?**

$ docker info

**11. How will you create a container in Docker in demonized (detached) mode?**

Docker run is the command to create a container using docker image. Below is the command to create container with default options.

#docker run -d –name=container\_name Image\_name (-d: demonized/detached mode)

**12. How will you create a container an connect to it via a shell (bash) prompt?**

#docker run -it –name=container\_name Image\_name /bin/bash (-d: demonized/detached mode)

**13. How to stop a container?**

$ docker container stop <container-hash-id>

**14. How to shutdown a container?**

$ docker container kill <container-hash-id>

**15. How to remove a container from host?**

$ docker container rm <container-hash-id>

**16. Can we lose our data when a Docker Container exits?**

No we don’t loose data if container exits, if no docker volumes are used then we will loose data if container is deleted.

**17. Can we run more than one process in a Docker container?**

It is generally recommended that you separate areas of concern by using one service per container. That service may fork into multiple processes, it’s ok to have multiple processes, but to get the most benefit out of Docker, avoid one container being responsible for multiple aspects of your overall application. You can connect multiple containers using user-defined networks and shared volumes.

Supervisor is a client/server system that allows its users to monitor and control a number of processes on UNIX-like operating systems.

**18. Can i use json instead of  yaml for my compose file?**

Yes. Yaml is a superset of json so any JSON file should be valid Yaml. To use a JSON file with Compose, specify the filename to use.

for example:

docker-compose -f docker-compose.json up

**19. What is Docker Swarm?**

A swarm is a group of machines that are running Docker and joined into a cluster.. The machines in a swarm can be physical or virtual. After joining a swarm, they are referred to as nodes.One of the key advantages of swarm services over standalone containers is that you can modify a service’s configuration, including the networks and volumes it is connected to, without the need to manually restart the service

**20. How do I change the Docker image installation directory?**

Edit file  /etc/docker/daemon.json and put “data-root=/to/user/dir”

**21. What is Docker Machine?**

Docker Machine is a tool that lets you install Docker Engine on virtual hosts, and manage the hosts with docker-machine commands. Machine to create Docker hosts on your local Mac or Windows box.

**22. How to link containers with each other ?.**

For linking a container with each other we need to start containers with –link option with followed by container name to which it has to be linked.

**23. What are dangling images and how to list them?.**

Dangling images are layers that have no relationship to any tagged images. They no longer serve a purpose and consume disk space. There’s no harm to delete the dangling images. Dangling images can be list by.

$ docker images -f dangling=true

**24. What are dangling containers and how to list them?**

Exited containers are treated as dangling containers. There’s no harm to delete the dangling containers. Use below command to list dangling containers:

$ docker ps -a -f status=exited

**25. What are dangling volumes and how to list them?**

When a volume exists and is no longer connected to any containers, it’s called a dangling volume. There’s no harm to delete the dangling volumes. Use below command to list dangling volumes:

$ docker volume ls -f dangling=true

**26. You are trying to start a container, but as soon as you start it it stops, How would check why it is getting exited?**

We can check logs of the container to find out why it got stopped by below command.

$ docker container logs -f container\_ID

**27. Suppose a container is running in background, how would you bring it to foreground.**

We can attach a container running in background by below command.

$ docker attach container\_name

**28. What is COW ?**

Copy-On-Write is a strategy of sharing and copying files for maximum efficiency. If a file or directory exists in a lower layer within the image, and another layer needs read access to it, it just uses the existing file. The first time another layer needs to modify the file,the file is copied into that layer and modified. This minimizes I/O and the size of each of the subsequent layers.

**29. How to find IP address of a running container ?**

We access any service running inside container with the host ip followed by the port number through which that container is bind. But we can check what ip container has taken by below command

$ docker container inspect <container-id> | grep -i ipaddress

**30. How to check size of running containers ?**

We can check size of any running container by below command .

$ docker ps -a -s

**31. What is “virtual size” of containers** ?

The amount of data used for the read-only image data used by the container plus the container’s writable layer size. Multiple containers may share some or all read-only image data. Two containers started from the same image share 100% of the read-only data.

**32. What is Dockerfile?**

A Dockerfile is a text document that contains all the commands a user could call on the command line to build an image. Below is the sample dockerfile for a container having python installed.

$ vi dockerfile

# Use an official Python runtime as a parent image  
FROM python:2.7-slim

# Set the working directory to /app  
WORKDIR /app

# Copy the current directory contents into the container at /app  
COPY . /app

# Install any needed packages specified in requirements.txt  
RUN pip install –trusted-host pypi.python.org

# Make port 80 available to the world outside this container  
EXPOSE 80

# Define environment variable  
ENV NAME World

# Run app.py when the container launches  
CMD [“python”, “app.py”]

**33. What are layers in docker images ?**

A Docker image is built up from a series of layers. Each layer represents an instruction in the image’s Dockerfile. Each layer except the very last one is read-only.

**34. What is the command to build an image from Dockerfile?**

$ docker build -t <image-name>:<tag> <locationOfDockerfile>

$ docker build -t friendlyPython .

“.” represent current directory in which Dockefile is kept

**35. What is docker swarm?**

A swarm is a group of machines that are running Docker and joined into a cluster. After that has happened, you continue to run the Docker commands you’re used to, but now they are executed on a cluster by a swarm manager. The machines in a swarm can be physical or virtual. After joining a swarm, they are referred to as nodes.

Swarm managers can use one of below strategies to run containers.  
emptiest node: which fills the least utilized machines with containers.  
Global: which ensures that each machine gets exactly one instance of the specified container.  
You instruct the swarm manager to use these strategies in the Compose file, just like the one you have already been using.

Swarm managers are the only machines in a swarm that can execute your commands, or authorize other machines to join the swarm as workers.  
Workers are just there to provide capacity and do not have the authority to tell any other machine what it can and cannot do.

**36. What is service in docker?.**

Services are really just “containers in production.” A service only runs one image, but it codifies the way that image runs—what ports it should use, how many replicas of the container should run so the service has the capacity it needs, and so on. Scaling a service changes the number of container instances running that piece of software, assigning more computing resources to the service in the process.

**37. How would you create a service in swarm?**

A service can be created by command

$ docker service create –name <service-name> <image-name>

**38. What is docker-compose file?**

A docker-compose.yml file is a YAML file that defines how Docker containers should behave in production. Below is the example of a docker-compose file.

$vi docker.compose.yml

version: “3”  
services:  
web:  
#replace username/repo:tag with your name and image details  
image: username/repo:tag  
deploy:  
replicas: 5  
restart\_policy:  
condition: on-failure  
resources:  
limits:  
cpus: “0.1”  
memory: 50M  
ports:  
– “80:80”  
networks:  
– webnet  
visualizer:  
image: dockersamples/visualizer:stable  
ports:  
– “8080:8080”  
volumes:  
– “/var/run/docker.sock:/var/run/docker.sock”  
deploy:  
placement:  
constraints: [node.role == manager]  
networks:  
– webnet  
networks:  
webnet:

**39. How to initialize swarm and deploy a service stack?**

$docker swarm init (It generates a token hash which is used by other nodes to join this swarm cluster)

$ docker stack deploy -c docker-compose.yml <stack-name>

**40. How to list docker stack?**

$ docker stack ls

**41. How to remove docker stack and taking all services down?**

$ docker stack rm <stack-name>

**42. How to identify all services in docker stack?**

$ docker stack ps <stack-name>

**43. How to identify all containers (tasks) in docker service?**

$ docker service ps <service-name-InStack>

**44. How to publish a port on running service ?**

$ docker service update –publish-add 8080 service\_name

**45. How to list and inspect all services?**

$docker service ls

$docker service inspect <service-name>

**46. Suppose you have do some maintenance on one of a node which is part of swarm cluster, How would you stop swarm service on that node ?**

Lets say we want to do some maintenance on node1 , we can drain that node fro swarm cluster and perform maintenance task below is the command to drain node from swarm cluster.

$ docker node update –availability drain node1

**47. How would you promote a normal node to be a manager node in swarm ?**

It can be done by below command.

$ docker node promote node\_name

**48. How can you join a node to swarm cluster?**

$ docker swarm join <swarm-token-hash>

**49. How can you leave a node from swarm cluster?**

Node can leave swarm by running below command

$ docker swarm leave (For worker node)

$ docker swarm leave –force (For master node)

**50. What are the different way to make the persistent data in docker container?**

Container data is lost one the container is terminated. To persist the container data we should use one of the below techniques.

**Volumes** are stored in a part of the host filesystem which is *managed by Docker* (/var/lib/docker/volumes/ on Linux). Non-Docker processes should not modify this part of the file system. Volumes are the best way to persist data in Docker. Example: Start ngnix container with a volume.

$docker run -d –name myapp –mount type=volume,source=myvol,target=/app ngnix:latest

**Bind mounts** may be stored *anywhere* on the host system. They may even be important system files or directories. Non-Docker processes on the Docker host or a Docker container can modify them at any time. Example: Start ngnix container with bind mount.

$docker run -d –mount type=bind,source=/techieglobus/myvol,target=/app nginx:latest

**tmpfs mounts** are stored in the host system’s memory only, and are never written to the host system’s filesystem. Example: Use a tmpfs mount in ngnix container.

$docker run -d –name myapp –mount type=tmpfs,destination=/app nginx:latest

**51. What are storage drivers in Docker?**

Docker storage drivers help to organize and manage containers data in background.

Docker storage drivers come as plugable modules. These modules can be installed, enabled, disabled and deleted at any point of time. We lose containers’ inside data when we change the storage driver types on running host.

Example: Select overlay2 as docker storage driver.

#service docker stop

#echo ‘{ “storage-driver”: “overlay2” }’ >> /etc/docker/daemon.json

#service docker start

**52. What are the native network drivers in Docker?**

* brigde (Default network name: bridge)
* host (Default network name: host)
* null (Default network name: none)
* overlay (Default network name: ingress)
* macvlan (Default network name: macvlan)

**53. What are the use cases of bridge network drivers and command to create it?**BRIDGE: It provides single host networking where all containers on same bridge network on same host can communicate to each other. It acts as virtual switch.

#docker network create -d bridge

**54. What are the use cases of bridge network drivers and command to create it?**

HOST: When you need to assign host’s IP to container. Container directly listens to on host port.

#docker network create -d host

**55. What are the use cases of bridge network drivers and command to create it?**

NULL: When you need to isolate your container from any network. No network communication to container.

#docker network create -d null

**56. What are the use cases of bridge network drivers and command to create it?**

OVERLAY: It provides native out-of-box multi-host networking. Generally used in cluster. It uses VXLAN technology and connect VXLAN tunnel between VTEP points on each host.

#docker network create -d overlay

**57. What are the use cases of bridge network drivers and command to create it?**

MACVLAN: It allows containers to participate directly in existing networks and VLANs. Promiscuous mode should be set on host’s ethernet card. It is used least.

#docker network create -d macvlan