**Dockers and Containers Question Bank**

**1.Discuss the problems solved by using the dockers and containers.**



**2.Discuss how containers differ from Hypervisor based virtualization.**

Containers also provide a way to isolate applications and provide a virtual platform for applications to run on *(see figure, b)*. Two main differences exist between a container and a hypervisor system.

The container’s system requires an underlying operating system that provides the basic services to all of the containerized applications using virtual-memory support for isolation. A hypervisor, on the other hand, runs VMs that have their own operating system using hardware VM support. Container systems have a lower overhead than VMs and container systems typically target environments where thousands of containers are in play. Container systems usually provide service isolation between containers. As a result, container services such as file systems or network support can have limited resource access.

There is also something called para-virtualization, which is sort of a mix between the two approaches. It uses virtual-memory support for isolation, but it requires special device drivers in the VM that are linked through the hypervisor to the underlying operating system, which in turn provides the device services.

Use cases:

Containers :

* Microservices
* Web applications
* DevOps testing
* Maximization of the amount of apps you can deploy per server

virtual machines :

* Running multiple applications together
* Monolithic applications
* Complete logical isolation between apps
* Legacy apps that require old operating systems

**3. Describe what difference does Docker bring to Containers.**

<https://drive.google.com/file/d/14VRrdUOGW_sgPSp26zuOI3_DbhisANaM/view>

**4.Illustrate with a diagram differences between container and virtual machine**



| 1 | VM is piece of software that allows you to install other software inside of it so you basically control it virtually as opposed to installing the software directly on the computer. | While a container is a software that allows different functionalities of an application independently. |
| --- | --- | --- |
| 2. | Applications running on VM system can run different OS. | While applications running in a container environment share a single OS. |
| 3. | VM virtualizes the computer system. | While containers virtualize the operating system only. |
| 4. | VM size is very large. | While the size of container is very light; i.e. a few megabytes. |
| 5. | VM takes minutes to run, due to large size. | While containers take a few seconds to run. |
| 6. | VM uses a lot of system memory. | While containers require very less memory. |
| 7. | VM is more secure. | While containers are less secure. |
| 8. | VM’s are useful when we require all of OS resources to run various applications. | While containers are useful when we are required to maximise the running applications using minimal servers. |
| 9. | Examples of VM are: KVM, Xen, VMware. | While examples of containers are:RancherOS, PhotonOS, Containers by Docker. |

**5-Differentiate between process, virtual machine and containers.**

****

**6. With a neat diagram explain the architecture of dockers.**

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.

**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**

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**

Docker Desktop is an easy-to-install application for your Mac or Windows 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.

**Docker registries**

A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.

**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.

**Images**

An image is a read-only template with instructions for creating a Docker container. Docker images act as a set of instructions to build a Docker container, like a template.A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run.

**Containers**

**A container** is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

l

**7. Define the following terms:**

**i) Image ii) Container iii) Dockerfile iv)Docker Client v) Docker Daemon/Engine**

i)Refer above ans

ii)Refer above ans

iii) **Dockerfile**

A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using `**docker build**` users can create an automated build that executes several command-line instructions in succession.Docker can build images automatically by reading the instructions from a **Dockerfile**

iv) Refer above ans

v) Refer above ans

**8. List out the similarities and differences of docker containers between Linux and Windows operating systems**

**Similarities**

1. Both are application containers, run natively, and do not depend on hypervisors or virtual machines.
2. Both administered through Docker CLI/APIs
3. They provide the same portability and modularity features on both operating systems.

**Differences**

1. Docker supports only Windows Server 2016 and Windows 10 now. But Docker can run on any type of modern Linux-based operating system.
2. Most container orchestration systems used for Docker on Linux are not supported on Windows.Only Docker Swarm is supported. Windows support for orchestrators such as Kubernetes and Apache Mesos is under development.

**9. With a neat diagram describe Docker Container Life Cycle.**

****

**OR**

****

**(Explain each step in short)**

**10. Illustrate basic workflow of the docker with a suitable diagram**

**Docker Workflow - Basics**

****

**11.Illustrate workflow of the docker with App Updates / Changes with a suitable**

**diagram**

****

**12.Describe different types of networks available for docker and containers.**

Containers provide services and to be able to access these services from outside the container, one would need to understand how the container exposes its service. Containers may also be distributed across many nodes. In such cases, knowledge of the available networks would allow you to choose the best network that suits your need

**Types of networks**

1. **None**

Used when your container doesn’t provide a service over the network

The container isn’t provided with an n/w interface and an IP address

docker run -d --network none --name testApp nginx

1. **Host**

Used when you would like to use the Docker host’s network stack

Services running on any port within your container will be directly accessible through the host’s IP

docker run -d --network host --name nginx\_on\_host nginx

1. **Bridge - default**

Used when your container doesn’t provide a service on the network

Needs “linking” for container DNS resolution

docker run -d --network bridge --name nginx\_bridge nginx

docker run -d --network bridge -p 8080:80 --name nginx\_on\_bridge\_with\_port nginx

1. **Bridge – user defined**

Used when you have a requirement for multiple networks on your Docker host, possibly because you’d like to isolate different deployments of containers on that host

docker network create --driver bridge isolated\_nw01

docker run -d --network isolated\_nw01 -p 8080:80 --name nginx\_ud\_bridge nginx

1. **Overlay (swarm) - user defined**

Multihost networking across Docker hosts that are part of the swarm

1. **Overlay (external key-value mechanism) – user defined**
2. Multihost networking across Docker hosts that are part of the swarm
3. In /lib/systemd/system/docker.service, add:

-H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock --cluster-store=consul://<consul\_IP>:8500 --cluster-advertise=<node\_IP>:2385

1. Reload options: `**systemctl daemon-reload`**
2. You can now create and use overlay networks:

All nodes: docker network create --driver overlay mhn01

Node 1: docker run -itd --network mhn01 --name=container1 busybox

Node 2: docker run -itd --network mhn01 --name=container2 busybox

**7. Custom network plugin**

In case you find the current networking options don’t fit your need, you could write your own network plugin using the Docker plugin API

**13.Explain the various ways in which a user can configure containers to be accessible**

1. --network

Ask Docker to connect your container to a specific network stack

1. "-p“

Ask Docker to “publish” specific ports on your container to the host’s ports

1. "-P“

Ask Docker to publish “All” the container’s active ports to the host

1. --link

Old way of establishing DNS registration for created containers in new containers. Still relevant for default bridge network.

op

**14. Show how a user can identify which network mode is being used by a container.**

1. docker inspect <container>, check under Object[‘NetworkSettings’][‘Networks’]

****

1. docker network ls
2. docker network inspect <network\_name>

**15. Explain why does a docker need a Union File System**

Union file system explanation - <https://www.terriblecode.com/blog/how-docker-images-work-union-file-systems-for-dummies/>

(Ans from stackoverflow)

It is used to:

* avoid duplicating a complete set of files each time you run an image as a new container
* isolate changes to a container filesystem in its own layer, allowing for that same container to be restarted from a known content (since the layer with the changes will have been dismissed when the container is removed)
* If you didn't have UnionFS, a 200MB image run 5 times as 5 separate

containers would mean 1GB of disk space.

**16. Discuss the types of docker mount with a neat diagram**

**Types**

1. **Volumes**
   1. Created and managed by Docker
      1. Explicitly (docker volume create)
      2. Implicitly (during container creation)
   2. Can be Named or Anonymous
   3. Supports volume drivers
   4. Cleanup possible (docker volume prune)
   5. Non docker processes cannot modify this part of the file system

.

1. **Bind Mounts**
   1. Created by docker if needed
   2. Maintained by host
   3. Can have security implications
   4. Performant
   5. Needs specific dir structure on host
   6. Non docker processes can modify this part of the file system.
2. **tmpfs Mounts**
   1. Only in memory! Never written to host’s file system
   2. Data is non persistent i.e. data is lost once the container stops running.
   3. Great for storing sensitive data
   4. Extremely fast
   5. No cleanup needed

Below image from Docker website shows all these 3 mounts and depicts where the data lives on the Docker host.



**17. Describe the characteristics and use cases of volumes**

1. Persistency across container lifecycle
2. Sharing data across containers (say across 2 nginx’s)
3. Decoupled host-container file system architecture
4. Storing data outside the host (central storage or cloud)
5. Backup and restore

**18. Describe the characteristics and use cases of bind mounts**

1. Sharing config from host
2. When the file or directory structure of the host machine is guaranteed to be consistent with the bind mounts the container requires.
3. DevOps build lifecycle – target folder into container
4. Persistency across container lifecycle
5. Sharing data across containers (say across 2 nginx’s)

**19.Describe the characteristics and use cases of Tmpfs mounts**

1. I/O sensitive projects
2. Standalone containers with need to store runtime info

**(Refer above ans for characteristics)**