Mayank Singh

Basic Docker Questions

1. **What is Docker?**
   Answer: Docker is an open-source platform that simplifies the development, shipping, and deployment of applications by using containers. Containers package applications and their dependencies, allowing them to run consistently across different environments. This ensures compatibility and eliminates issues related to environment discrepancies.
2. **What is a Docker container?**
   Answer: A Docker container is a lightweight, isolated environment that runs applications with all the necessary dependencies. It shares the host OS kernel, which makes it efficient and fast compared to virtual machines. Containers can be easily started, stopped, or replicated, providing flexibility and scalability for applications.
3. **What is Docker Hub?**
   Answer: Docker Hub is a cloud-based registry where Docker users can store, share, and download container images. It provides access to a large collection of official and user-created images, making it easy to find pre-configured containers for common applications like databases, web servers, and more.
4. **What is a Docker image?**
   Answer: A Docker image is a read-only template used to create containers. Images contain the application code, dependencies, and an OS layer required to run the application. You can build images from scratch or pull them from registries like Docker Hub. Each image has multiple layers, making it efficient by only adding what's needed.

5. **How to pull an image from Docker Hub?**
   Answer: To pull an image, use the docker pull <image-name> command, which downloads the image from Docker Hub to your local machine. For example, docker pull nginx pulls the latest NGINX server image, making it ready for container creation on your system.

6. **How to list all Docker containers?**
   Answer: Use docker ps to view running containers or docker ps -a to view all containers, including stopped ones. The output provides details such as container ID, image name, status, and ports, helping you manage and track the containers on your system.

7. **How to start a Docker container?**
   Answer: To start a container, use docker start <container-id>. This command resumes a previously created container without running it from scratch, preserving its state and data. You can use either the container ID or name to identify it.

8. **How to stop a Docker container?**
   Answer: Use docker stop <container-id> to gracefully stop a running container. This command sends a SIGTERM signal, giving the container time to shut down. If it doesn’t respond, Docker may follow up with a SIGKILL signal to forcibly terminate the process.

9. **How to remove a Docker container?**
   Answer: To remove a container, use docker rm <container-id>. Only stopped containers can be removed, so you may need to stop it first. You can remove multiple containers simultaneously by providing their IDs or using the -f flag to force-remove running containers.

10. **How to remove a Docker image?**
    Answer: Use docker rmi <image-name> to delete an image

from your local repository. This helps free up disk space. However, if any containers are using the image, Docker will not allow deletion unless you remove those containers first.

11. **How to create a Docker container from an image?**
Answer: Use docker run <image-name> to create and start a container from an image. You can add options like -d to run in detached mode, -p to map ports, or --name to give the container a custom name, making it easier to manage.

12. **What is the difference between docker run and docker start?**
Answer: docker run creates and starts a new container, whereas docker start only starts a stopped container. With docker run, the container is initialized with a fresh environment, while docker start resumes the container in its previous state.

13. **How to view the logs of a container?**
Answer: Use docker logs <container-id> to display a container's logs. You can view application output, errors, and runtime information. Options like -f for following live logs and --tail for showing recent logs are helpful for debugging and monitoring.

14. **How to execute a command inside a running container?**
Answer: Use docker exec -it <container-id> <command> to run commands inside an active container. For example, docker exec -it my_container bash opens a shell session in the container, allowing you to inspect and manage it directly.

15. **How to build a Docker image from a Dockerfile?**
Answer: Use docker build -t <image-name> . to create an image from a Dockerfile in the current directory. The -t flag tags the image, making it easier to reference later. Docker reads instructions in the Dockerfile to assemble the image layer by layer.

16. **What is a Dockerfile?**
Answer: A Dockerfile is a text file with a series of instructions for building a Docker image. Commands like FROM, COPY, and RUN define the base image, copy files, and execute commands, respectively. Docker interprets each line to create an image layer by layer.

17. **What is the purpose of the FROM instruction in a Dockerfile?**
Answer: FROM specifies the base image for the Docker image. It's the first line in a Dockerfile and sets the foundation upon which other layers are built. For example, FROM ubuntu:latest starts with the latest Ubuntu image.

18. **What is the COPY instruction in a Dockerfile?**
Answer: COPY transfers files or directories from the host system to the container's filesystem. It's used to include source code, configuration files, or other assets. For example, COPY . /app copies all files from the current directory to the /app folder in the container.

19. **What is the EXPOSE instruction in a Dockerfile?**
Answer: EXPOSE specifies which ports the container listens on at runtime. For example, EXPOSE 80 indicates that the container's service is available on port 80. This does not publish the port; it just hints to users of the container.

20. **What is the difference between COPY and ADD in a Dockerfile?**
Answer: While COPY only transfers local files, ADD can handle remote URLs and automatically extracts compressed files. However, COPY is generally preferred for its simplicity and clearer purpose unless extra functionality is required.

21. **How to check the size of a Docker image?**
Answer: Use docker images to view a list of images along with

their sizes. The SIZE column shows the total disk space each image occupies, which helps you monitor and optimize image storage.

22. **What is Docker Compose?**
Answer: Docker Compose is a tool that enables defining and running multi-container Docker applications. You use a YAML file (docker-compose.yml) to configure your application's services, networks, and volumes, making it easy to start all services with a single command.

23. **How to define a Docker Compose file?**
Answer: A Docker Compose file, docker-compose.yml, defines services, networks, and volumes. Each service is a container with specified settings, such as ports, environment variables, and dependencies, allowing multi-container applications to be orchestrated effortlessly.

24. **How to start services with Docker Compose?**
Answer: Use docker-compose up to start all services defined in docker-compose.yml. The -d flag can be added to run services in detached mode. Compose builds images, creates containers, and starts them automatically.

25. **How to stop services with Docker Compose?**
Answer: Use docker-compose down to stop and remove all running services, networks, and volumes created by docker-compose up. It cleans up resources and brings down the application stack smoothly.

26. **How to scale services in Docker Compose?**
Answer: Use docker-compose up --scale <service>=<number> to specify the number of containers for a service. For example, docker-compose up --scale web=3 runs three instances of the web service, useful for load balancing and testing distributed applications.

27. • **What is the `ENTRYPOINT` instruction in a Dockerfile?**
Answer: `ENTRYPOINT` specifies the main command that runs when a container starts, allowing for fixed commands with parameters passed during `docker run`. For example, `ENTRYPOINT ["python"]` makes `python` the default executable, which can then run specific scripts as arguments.

28. • **What is the difference between `ENTRYPOINT` and `CMD`?**
Answer: `ENTRYPOINT` sets the primary command, while `CMD` provides default arguments. `ENTRYPOINT` is for core tasks that don't change, and `CMD` can override or extend it. For instance, `ENTRYPOINT ["python"]` and `CMD ["script.py"]` would run `python script.py` by default.

29. • **How do you share data between Docker containers?**
Answer: Data sharing between containers is done via **volumes** or **bind mounts**. Volumes are managed by Docker, while bind mounts map host directories directly. You can also use Docker networks to link containers for data sharing through network calls.

30. • **What are Docker volumes, and why use them?**
Answer: Volumes are storage areas managed by Docker outside the container's filesystem, preserving data across restarts. They are ideal for sharing data between containers and persisting files like logs, databases, and configuration files.

31. • **How do you create and attach a volume to a container?**
Answer: First, create a volume using `docker volume create <volume-name>`. Then, use `docker run -v <volume-name>:/container-path <image>` to attach it. This setup allows the container to read and write files to the volume path.

32. • **What is Docker Swarm?**
Answer: Docker Swarm is Docker's native orchestration tool, allowing clusters of Docker nodes to act as a single system. It

manages container scheduling, load balancing, and scaling, ideal for deploying large applications across multiple hosts.

33. • **What are the key features of Docker Swarm?**
Answer: Key features include service scaling, load balancing, high availability, and secure networking. Swarm uses nodes as managers and workers, distributing services across them to ensure fault tolerance and resilience.

34. • **How to create a Docker network?**
Answer: Use `docker network create <network-name>` to create a network for connecting containers. Containers on the same network can communicate easily. Docker networks allow multi-container applications to interact securely and effectively.

35. • **What is a bridge network in Docker?**
Answer: A bridge network is the default network type that Docker creates. Containers within the same bridge network can communicate with each other via their IPs or hostnames, but they are isolated from the external network unless ports are published.

36. • **What is the purpose of Docker Compose's `depends_on`?**
Answer: `depends_on` in Docker Compose specifies the order in which services start. For instance, `depends_on: - db` ensures that a service waits until the `db` service is started. Note that it doesn't guarantee readiness, only startup sequence.

37. • **How can you check the Docker version?**
Answer: Use `docker --version` for a concise version, or `docker version` for detailed output about client and server versions, build details, and API compatibility. This helps ensure you're working with the correct version of Docker.

38. • **How to restart a container automatically if it crashes?**
Answer: Use the `--restart` option with values like `always` or `on-failure` when running a container, e.g., `docker run --restart=always <image>`. This ensures the

container restarts automatically upon failure, enhancing service reliability.

39. • **How can you limit a container's memory and CPU usage?**
Answer: Use `--memory` and `--cpus` flags to restrict resource usage. For example, `docker run --memory=512m --cpus=1 <image>` limits the container to 512MB of memory and 1 CPU core. This helps in managing host resources efficiently.

40. • **What is the Docker Daemon?**
Answer: The Docker Daemon (`dockerd`) runs on the host system and manages Docker objects like containers, images, and networks. It listens to API requests and executes commands, acting as the core engine for container management.

41. • **How to inspect a Docker container's configuration?**
Answer: Use `docker inspect <container-id>` to retrieve JSON-formatted configuration details, including environment variables, IP addresses, mounts, and network settings, providing in-depth information for troubleshooting or analysis.

42. • **What is the difference between `docker-compose up` and `docker-compose start`?**
Answer: `docker-compose up` creates and starts services based on the YAML file, while `docker-compose start` only starts services that are already created. Use `up` for the initial start and `start` for subsequent restarts.

43. • **What are Docker tags?**
Answer: Tags are labels for Docker images, typically specifying versions. For instance, `nginx:latest` or `nginx:1.19.6`. Tags help differentiate between versions of the same image, allowing easier management and consistent deployments.

44. • **How to tag a Docker image?**
Answer: Use `docker tag <image-id> <repository>:<tag>`. For example, `docker tag nginx my-nginx:v1.0` assigns the `v1.0` tag to the `my-`

`nginx` image, making it identifiable for specific versions or purposes.

45. • **What is multi-stage building in Docker?**
Answer: Multi-stage builds allow creating optimized images by using intermediate stages to compile or configure. Each stage can discard unnecessary files, reducing the final image size. Multi-stage builds are useful in minimizing production image footprints.

46. • **How to connect a Docker container to multiple networks?**
Answer: Use `docker network connect <network-name> <container-id>` to add a container to an additional network. This allows containers to communicate with multiple services, useful for complex, multi-container setups.

47. • **What is `docker stats`, and how does it help?**
Answer: `docker stats` displays real-time resource usage of containers, including CPU, memory, network, and I/O stats. It helps monitor performance and identify resource bottlenecks for effective troubleshooting and optimization.

48. • **How to push an image to Docker Hub?**
Answer: First, tag the image with your Docker Hub username and repository (e.g., `docker tag nginx myuser/nginx:v1.0`), then push it using `docker push myuser/nginx:v1.0`. This makes the image available for public or private access on Docker Hub.

49. • **What is the purpose of `.dockerignore`?**
Answer: `.dockerignore` is a file that specifies files or directories to exclude from the Docker image. It reduces image size and build time by ignoring unnecessary files, like `node_modules` or temporary files, improving efficiency.

50. • **How to remove unused Docker resources?**
Answer: Use `docker system prune` to delete unused containers, images, volumes, and networks. For a more thorough cleanup, `docker system prune -a` removes all unused images, helping manage storage effectively.

51. • **What is Kubernetes, and how does it relate to Docker?**
Answer: Kubernetes is an open-source orchestration platform for automating deployment, scaling, and management of containerized applications. While Docker handles individual containers, Kubernetes manages clusters of containers, providing robust support for complex, distributed applications.