50 Docker Interview Questions

1. **What is Docker?**
    - Docker is a containerization platform that enables developers to package, distribute, and run applications in isolated environments called containers.
2. **Explain the difference between a container and a virtual machine (VM).**
    - Containers share the host OS kernel, making them lightweight and more efficient compared to VMs, which require a full OS for each instance.
3. **How does Docker work?**
    - Docker uses containerization technology to package applications and their dependencies into containers. Containers share the host OS kernel, ensuring consistency across different environments.
4. **What is a Docker image?**
    - A Docker image is a lightweight, standalone, and executable package that includes the application code, libraries, dependencies, and runtime needed to run an application.
5. **What is a Docker container?**
    - A Docker container is a runnable instance of a Docker image. It encapsulates the application and its dependencies in an isolated environment.
6. **Explain the role of Docker Engine.**
    - Docker Engine is the core component responsible for building, running, and managing Docker containers. It includes a server, API, and a command-line interface.

Docker Commands:

7. **What is the command to pull a Docker image from Docker Hub?**
    - `docker pull <image_name>:<tag>`
8. **How to run a Docker container?**
    - `docker run <image_name>`
9. **Explain the purpose of `docker ps` command.**
    - `docker ps` lists the currently running containers, providing information such as container ID, image used, and status.
10. **How to stop a running Docker container?**
    - `docker stop <container_id>`
11. **What is the difference between `docker rm` and `docker rmi`?**
    - `docker rm` removes stopped containers, and `docker rmi` removes Docker images.

Dockerfile:

12. **What is a Dockerfile?**
    - A Dockerfile is a script that contains instructions for building a Docker image.
13. **Explain the significance of the `ENTRYPOINT` and `CMD` instructions in a Dockerfile.**
    - `ENTRYPOINT` sets the command to be executed when the container starts, and `CMD` provides default arguments for the entry point.
14. **How to build a Docker image from a Dockerfile?**
    - `docker build -t <image_name>:<tag> <path_to_Dockerfile>`
15. **What is the purpose of the `.dockerignore` file?**
    - `.dockerignore` specifies files and directories to exclude when building a Docker image.

Docker Networking:

16. **Explain the default network mode in Docker.**
    - The default network mode is bridge, which allows containers on the same host to communicate.

17. **How to expose ports in a Docker container?**
    - Use the `-p` or `--publish` flag when running a container: `docker run -p <host_port>:<container_port>`
18. **What is Docker Compose, and why would you use it?**
    - Docker Compose is a tool for defining and running multi-container Docker applications. It simplifies the process of defining and orchestrating containers.

Docker Volumes:

19. **Why are Docker volumes used?**
    - Docker volumes provide persistent storage for containers, allowing data to persist even if the container is stopped or removed.
20. **How to create a Docker volume?**
    - `docker volume create <volume_name>`
21. **Explain the difference between a bind mount and a Docker volume.**
    - A bind mount links a directory on the host with a directory in the container, while a Docker volume is a managed storage solution created and maintained by Docker.

Docker Swarm:

22. **What is Docker Swarm?**
    - Docker Swarm is a native clustering and orchestration solution for Docker. It allows you to create and manage a swarm of Docker nodes.
23. **How to initialize a Docker Swarm?**
    - `docker swarm init`
24. **Explain the role of managers and workers in Docker Swarm.**
    - Managers control the swarm and orchestrate tasks, while workers execute the tasks assigned by the managers.

Docker Security:

25. **How can you improve Docker container security?**
    - Regularly update Docker and its components, use official images, minimize the number of running processes, and apply the principle of least privilege.
26. **What is the purpose of Docker Content Trust?**
    - Docker Content Trust ensures the integrity and authenticity of images by signing them using digital signatures.

Docker Compose:

27. **How to define services in a Docker Compose file?**
    - Services are defined under the `services` key in a Docker Compose file.
28. **Explain the use of the `docker-compose up` command.**
    - `docker-compose up` starts the defined services in a Docker Compose file.
29. **How to scale services in Docker Compose?**
    - `docker-compose up --scale <service_name>=<num_instances>`

Docker Registry:

30. **What is a Docker Registry?**
    - A Docker Registry is a storage and distribution system for Docker images. Docker Hub is a popular public registry.
31. **How to push a Docker image to Docker Hub?**
    - `docker push <image_name>:<tag>`
32. **Explain the significance of the `docker login` command.**
    - `docker login` authenticates the Docker CLI with a registry, allowing you to push and pull images.

Docker Monitoring:

33. **How to check container logs in Docker?**
    - `docker logs <container_id>`

34. **Explain the purpose of the `docker stats` command.**
    - `docker stats` provides real-time resource usage statistics for running containers.

Docker Troubleshooting:

35. **What to do if a container exits immediately after starting?**
    - Use `docker logs <container_id>` to check the container logs for errors.
36. **How to remove all stopped containers?**
    - `docker container prune`

Docker and Microservices:

37. **How can Docker be beneficial in a microservices architecture?**
    - Docker provides lightweight and scalable containerization, making it easier to deploy and manage microservices independently.
38. **Explain the concept of service discovery in Docker Swarm.**
    - Service discovery in Docker Swarm allows containers to discover and communicate with each other by using service names.

Docker Best Practices:

39. **What are some best practices for creating Dockerfiles?**
    - Use official base images, minimize the number of layers, and clean up unnecessary artifacts in each step.
40. **How to manage secrets in Docker Swarm?**
    - Use Docker secrets to securely manage sensitive information, such as passwords or API keys.

Docker and Continuous Integration/Continuous Deployment (CI/CD):

41. **How does Docker fit into a CI/CD pipeline?**
    - Docker containers provide consistency across different environments, making it easier to build and deploy applications in a CI/CD pipeline.

42. **Explain the concept of blue-green deployment with Docker.**
    - Blue-green deployment involves deploying a new version of an application

alongside the existing one and switching traffic once the new version is deemed stable.

Docker and Kubernetes:

43. **What is Kubernetes, and how does it relate to Docker?**
    - Kubernetes is a container orchestration platform that can manage and scale Docker containers. It provides additional features for container deployment, scaling, and management.
44. **How to use Docker containers in a Kubernetes cluster?**
    - Kubernetes can deploy and manage Docker containers as part of its workloads.

Docker and Cloud Platforms:

45. **How is Docker used in cloud platforms like AWS, Azure, or GCP?**
    - Cloud platforms provide container orchestration services (e.g., AWS ECS, Azure Kubernetes Service) that simplify the deployment and management of Docker containers.
46. **What is Docker Swarm Mode, and how does it differ from standalone Docker Swarm?**
    - Docker Swarm Mode is an orchestration feature built into Docker Engine, providing native support for creating and managing swarms.

Docker Security Best Practices:

47. **How to secure Docker containers and images?**
    - Regularly update Docker and its dependencies, use secure base images, and implement network segmentation.

48. **Explain Docker container isolation.**
    - Containers are isolated from each other and the host system using namespaces and control groups (cgroups).

Docker and DevOps:

49. **How does Docker contribute to the DevOps workflow?**
    - Docker facilitates consistency in development, testing, and production environments, enabling seamless collaboration between development and operations teams.
50. **What are some challenges when working with Docker, and how can they be mitigated?**
    - Challenges may include security concerns, image size, and complexity in orchestration. Mitigate these by adopting security best practices, optimizing images, and using orchestration tools like Docker Swarm or Kubernetes.