DevOps Shack

Docker | Interview Questions With Answers

1. Docker Basics:

Question 1: What is Docker?

Answer: Docker is a platform for developing, shipping, and running applications in containers. It allows developers to package an application with all of its dependencies into a standardized unit for easy deployment.

Question 2: Explain the difference between Docker image and Docker container.

Answer: A Docker image is a lightweight, standalone, executable package that includes everything needed to run a piece of software, including the code, runtime, libraries, and dependencies. A Docker container is a runtime instance of a Docker image.

Question 3: How does Docker achieve lightweight virtualization?

Answer: Docker uses containerization technology, which leverages features of the Linux kernel (such as namespaces and cgroups) to create isolated environments called containers. These containers share the host operating system's kernel but run in separate user spaces, making them lightweight and efficient.

2. Docker Architecture:

Question 4: Explain the components of the Docker architecture.

Answer: The Docker architecture consists of the Docker Engine, Docker Images, Docker Containers, Docker Registry, and Docker Compose. The Docker Engine is the core component responsible for building and running Docker containers. Docker Images are read-only templates used to create Docker containers. Docker Containers are instances of Docker images that run applications. The Docker Registry is a repository for Docker images, and Docker Compose is a tool for defining and running multi-container Docker applications.

Question 5: What is Docker Engine?

Answer: Docker Engine is a client-server application with three major components: a server, a REST API, and a command-line interface (CLI). The Docker Engine server is a long-running process (daemon) that manages Docker containers and images on a host system.

Question 6: How does Docker Networking work?

Answer: Docker Networking allows containers to communicate with each other and with the outside world. Docker creates virtual networks that containers can be attached to, enabling communication via IP addresses and ports. Docker also supports various network drivers for different use cases, such as bridge, overlay, and host networking.

3. Docker Commands:

Question 7: What are some common Docker commands?

Answer: Some common Docker commands include `docker run` (to run a container), `docker build` (to build an image), `docker pull` (to pull an image from a registry), `docker push` (to push an image to a registry), `docker ps` (to list running containers), `docker images` (to list available images), `docker stop` (to stop a container), and `docker rm` (to remove a container).
Question 8: Explain the difference between `docker run` and `docker exec`.
Answer: `docker run` is used to create and start a new container from an image, while `docker exec` is used to execute commands inside a running container.
Question 9: How can you remove all Docker containers?

Answer: You can remove all Docker containers by running the command `docker rm $(docker ps -aq)`.

4. Dockerfile:

Question 10: What is a Dockerfile?

Answer: A Dockerfile is a text file that contains instructions for building a Docker image. It specifies the base image to use, the commands to run during the build process, and any configuration or metadata for the image.

Question 11: Explain the difference between `CMD` and `ENTRYPOINT` in a Dockerfile.
Answer: `CMD` is used to specify the default command to run when a container starts, while `ENTRYPOINT` is used to specify the executable that will run when the container starts. If both are present, `ENTRYPOINT` will be executed with `CMD` as its arguments.
Question 12: How can you pass arguments to a Dockerfile during the build process?

Answer: You can use build arguments in a Dockerfile by specifying them with the `ARG` instruction and passing values to them using the `--build-arg` flag with the `docker build` command.

5. Docker Networking:

Question 13: What is Docker bridge network?

Answer: Docker bridge network is the default network mode, which allows containers connected to the same bridge network to communicate with each other using IP addresses. It provides isolation and allows containers to access external networks through NAT (Network Address Translation).

Question 14: How can you create a custom bridge network in Docker?

Answer: You can create a custom bridge network using the `docker network create` command, specifying the `--driver bridge` option followed by the desired network name.
Question 15: Explain Docker overlay network.

Answer: Docker overlay network facilitates communication between containers across multiple Docker hosts. It uses VXLAN (Virtual Extensible LAN) technology to create an overlay network that spans across hosts, enabling containers to communicate seamlessly as if they were on the same host.

6. Docker Volumes:

Question 16: What is a Docker volume?

Answer: A Docker volume is a persistent data storage mechanism that allows data to persist beyond the lifetime of a container. Volumes are used to share data between containers or to persist data even if the container is stopped or deleted.

Question 17: How can you create a Docker volume?

Answer: You can create a Docker volume using the `docker volume create` command, followed by the desired volume name.
Question 18: Explain the difference between Docker bind mounts and volumes.

Answer: Docker bind mounts map a host file or directory to a container file or directory, allowing the container to access files from the host system. Volumes, on the other hand, are managed by Docker and provide persistent storage that can be shared among containers and persists even if the container is removed.

7. Docker Compose:

Question 19: What is Docker Compose?

Answer: Docker Compose is a tool for defining and running multi-container Docker applications. It uses YAML files to configure the application's services, networks, and volumes, making it easy to manage complex Docker setups.

Question 20: How do you define services in a Docker Compose file?

Answer: Services in a Docker Compose file are defined under the `services` section using YAML syntax, where each service specifies its image, ports, environment variables, volumes, and other configuration options.
Question 21: Explain the difference between `docker-compose up` and `docker-compose start`.
Answer: `docker-compose up` creates and starts containers for all services defined in the Docker Compose file, while `docker-compose start` starts existing containers that were previously created but stopped.
8. Docker Security:

Question 22: How can you improve Docker container security?

Answer: You can improve Docker container security by using official images from trusted sources, regularly updating images and containers, minimizing the attack surface by using the principle of least privilege, implementing network segmentation, using Docker Content Trust to verify image integrity, and monitoring container activity.

Question 23: What is Docker Content Trust (DCT)?

Answer: Docker Content Trust is a security feature that allows you to verify the authenticity and integrity of Docker images. When enabled, DCT uses cryptographic signatures to ensure that only trusted images are pulled and run on Docker hosts.

Question 24: How can you scan Docker images for vulnerabilities?

Answer: You can scan Docker images for vulnerabilities using tools like Docker Security Scanning, Clair, Trivy, or Anchore. These tools analyze the contents of Docker images and report any known vulnerabilities present in the image's dependencies.

9. Docker Swarm:

Question 25: What is Docker Swarm?

Answer: Docker Swarm is a clustering and orchestration tool provided by Docker for managing a cluster of Docker hosts. It enables you to deploy, scale, and manage containers across multiple hosts, providing high availability and load balancing for containerized applications.

Question 26: How do you initialize a Docker Swarm?

Answer: You can initialize a Docker Swarm by running the command `docker swarm init` on a Docker host, which initializes the host as a Swarm manager and generates a join token that other hosts can use to join the Swarm.
Question 27: Explain the difference between Docker Swarm mode and Docker standalone mode.

Answer: Docker Swarm mode is a clustering and orchestration feature built into Docker Engine, allowing you to create and manage a Swarm cluster for deploying and scaling containers across multiple hosts. Docker standalone mode refers to running Docker without clustering or orchestration features.

10. Docker Security:

Question 28: What is Docker Content Trust (DCT)?

Answer: Docker Content Trust is a security feature that allows you to verify the authenticity and integrity of Docker images. When enabled, DCT uses cryptographic signatures to ensure that only trusted images are pulled and run on Docker hosts.

Question 29: How can you improve Docker container security?

Answer: You can improve Docker container security by using official images from trusted sources, regularly updating images and containers, minimizing the attack surface by using the principle of least privilege, implementing network segmentation,

using Docker Content Trust to verify image integrity, and monitoring container activity.

Question 30: What are Docker security best practices?

Answer: Docker security best practices include using least privilege, keeping images and containers up to date, avoiding running containers as root, using Docker Content Trust, implementing network segmentation, scanning images for vulnerabilities, and monitoring container activity.

11. Docker Monitoring and Logging:

Question 31: How can you monitor Docker containers?

Answer: You can monitor Docker containers using tools like Docker Stats, Docker Events, cAdvisor, Prometheus with Grafana, and third-party monitoring solutions. These tools provide insights into container resource usage, performance metrics, and health status.

Question 32: What are some logging options available in Docker?

Answer: Docker provides logging drivers that allow you to control how container logs are handled. Some logging options include the `json-file` driver (default), `syslog`, `journald`, `gelf`, `fluentd`, `awslogs`, and `splunk`.
Question 33: How can you view container logs in Docker?

Answer: You can view container logs in Docker using the `docker logs` command followed by the container ID or name. Additionally, you can use logging drivers to redirect container logs to external logging systems for centralized log management.

12. Docker Image Management:

Question 34: What is Docker image caching?

Answer: Docker image caching is a mechanism used during the build process to speed up subsequent builds by reusing intermediate layers from previous builds. When a Dockerfile is built, each instruction produces a layer, and Docker caches these layers to avoid rebuilding them if the instruction and its context haven't changed.

Question 35: How can you list all Docker images on a system?

Answer: You can list all Docker images on a system using the `docker images` command, which displays a list of all images along with their repository, tag, and size.

Question 36: Explain Docker image layering.

Answer: Docker image layering is a concept where each instruction in a Dockerfile produces a layer in the resulting image. Layers are read-only and represent changes to the filesystem, such as adding files or executing commands. Docker uses Union file systems to combine these layers into a single image.

13. Docker Deployment Strategies:

Question 37: What are the different deployment strategies in Docker?

Answer: Different deployment strategies in Docker include rolling updates, blue-green deployments, canary deployments, and A/B testing. These strategies enable you to deploy application updates with minimal downtime and risk by gradually transitioning traffic to new versions.

Question 38: Explain rolling updates in Docker.

Answer: Rolling updates in Docker involve updating containers one at a time in a rolling fashion, where each new container version is deployed and verified before moving on to the next one. This strategy ensures that the application remains available during the update process.

Question 39: What is a blue-green deployment in Docker?

Answer: A blue-green deployment in Docker involves maintaining two identical production environments, one active (blue) and one inactive (green). Updates are deployed to the inactive environment, and traffic is switched from the blue to the green environment once the update is verified.

14. Docker Registry:

Question 40: What is Docker Registry?

Answer: Docker Registry is a service for storing and distributing Docker images. It can be either Docker Hub (public registry) or a private registry like Docker Trusted Registry (DTR) or self-hosted registry using Docker Registry open-source software.

Question 41: How can you push a Docker image to a registry?

Answer: You can push a Docker image to a registry using the `docker push` command followed by the image name and tag, along with the registry URL if it's not Docker Hub.

Question 42: What is Docker Hub?

Answer: Docker Hub is a cloud-based repository provided by Docker for storing and sharing Docker images. It hosts millions of public images and allows users to store and distribute their own images publicly or privately.

15. Docker in CI/CD Pipeline:

Question 43: How can you integrate Docker into a CI/CD pipeline?

Answer: Docker can be integrated into a CI/CD pipeline by using Docker images as build environments, running tests inside Docker containers, building Docker images as part of the pipeline, and deploying applications using Docker containers to various environments.

Question 44: Explain how Docker can improve CI/CD workflows.

Answer: Docker can improve CI/CD workflows by providing consistent build environments across different stages of the pipeline, enabling reproducible builds, speeding up build times with image caching, and simplifying deployment by encapsulating applications and dependencies into containers.

Question 45: What are some CI/CD tools that support Docker integration?

Answer: Some CI/CD tools that support Docker integration include Jenkins, GitLab CI/CD, Travis CI, CircleCI, TeamCity, and GitHub Actions. These tools provide native support for Docker, allowing you to build, test, and deploy applications using Docker containers.

16. Docker for Development:

Question 46: How can Docker improve the development workflow?

Answer: Docker can improve the development workflow by providing consistent environments across development, testing, and production, reducing environment setup time, enabling isolation of dependencies, facilitating collaboration among team members, and simplifying application deployment.

Question 47: What are some Docker development best practices?

Answer: Docker development best practices include using Docker Compose for defining multi-container applications, creating lightweight and single-purpose images, optimizing Dockerfile layers, using bind mounts or volumes for local development, and leveraging Docker layer caching.

Question 48: How do you debug Docker containers?

Answer: You can debug Docker containers by attaching to a running container with the `docker exec` command, inspecting container logs, using interactive terminals (`docker run -it`), or connecting to the container's network namespace to troubleshoot network issues.

17. Docker Orchestration:

Question 49: What is Docker orchestration?

Answer: Docker orchestration is the process of managing and coordinating the deployment, scaling, and operation of Docker containers across a cluster of hosts. It involves tasks such as load balancing, service discovery, scheduling, and health monitoring.

Question 50: What are some Docker orchestration tools?

Answer: Some popular Docker orchestration tools include Docker Swarm (built-in), Kubernetes, Apache Mesos, Amazon ECS (Elastic Container Service), and Google Kubernetes Engine (GKE). These tools provide features for automating container management and scaling.

Question 51: Explain the difference between Docker Swarm and Kubernetes.

Answer: Docker Swarm is a simple and easy-to-use container orchestration tool provided by Docker, while Kubernetes is a more feature-rich and complex orchestration platform originally developed by Google. Kubernetes offers advanced scheduling, service discovery, and scaling capabilities compared to Docker Swarm.

18. Docker High Availability:

Question 52: How can you achieve high availability with Docker Swarm?

Answer: High availability with Docker Swarm can be achieved by running multiple Swarm manager nodes for redundancy, enabling automatic service rescheduling, using load balancing for distributing traffic, and implementing health checks to detect and recover from container failures.

Question 53: What is Docker service scaling?

Answer: Docker service scaling refers to the ability to increase or decrease the number of replicas of a service running in a Docker Swarm cluster. Scaling allows you to distribute workload across multiple containers to handle varying levels of traffic and ensure high availability.

Question 54: How does Docker handle node failures in a Swarm cluster?

Answer: Docker Swarm handles node failures by automatically rescheduling failed containers on healthy nodes, based on the desired state defined in the service configuration. Swarm managers monitor the health of nodes and containers, and if a node becomes unavailable, Swarm reschedules the affected containers to maintain service availability.

19. Docker for Microservices:

Question 55: How can Docker facilitate microservices architecture?

Answer: Docker facilitates microservices architecture by providing lightweight and isolated containers for individual microservices, enabling seamless deployment, scaling, and management of microservices, allowing each microservice to be developed and tested independently, and simplifying service discovery and communication.

Question 56: What are some challenges of deploying microservices with Docker?

Answer: Some challenges of deploying microservices with Docker include managing container orchestration complexity, ensuring service discovery and communication between microservices, maintaining consistent environments across development, testing, and production, and managing container networking and security at scale.

Question 57: How can you deploy a multi-container microservices application with Docker Compose?

Answer: You can deploy a multi-container microservices application with Docker Compose by defining each microservice as a separate service in a Docker Compose file, specifying dependencies, networks, volumes, and environment variables, and then using `docker-compose up` to launch the entire application stack.

20. Docker Networking:

Question 58: What is Docker overlay network?

Answer: Docker overlay network is a networking mechanism that enables communication between containers deployed across multiple Docker hosts. It uses VXLAN (Virtual Extensible LAN) encapsulation to create a virtual network overlay that

spans across hosts, allowing containers to communicate as if they were on the same host.

Question 59: How can you create an overlay network in Docker Swarm?

Answer: You can create an overlay network in Docker Swarm using the `docker network create` command with the `--driver overlay` option, followed by the desired network name. This network will be available to all services and containers within the Swarm cluster.
Question 60: Explain Docker bridge network.

Answer: Docker bridge network is the default network mode in Docker, which creates a virtual bridge on the host system and assigns IP addresses to containers connected to that bridge. Containers on the same bridge network can communicate with each other using IP addresses, and the bridge network provides NAT (Network Address Translation) for external connectivity.

21. Docker Security:

Question 61: What is Docker Content Trust (DCT)?

Answer: Docker Content Trust (DCT) is a security feature that provides cryptographic verification of image authenticity and integrity. It ensures that only signed and trusted images are pulled and run on Docker hosts, preventing the execution of potentially malicious or tampered images.

Question 62: How can you enable Docker Content Trust?

Answer: You can enable Docker Content Trust by setting the `DOCKER_CONTENT_TRUST` environment variable to `1` or `true` on Docker clients, which enables image signing and verification. Additionally, Docker Content Trust can be enforced globally by configuring Docker daemon settings.
Question 63: What are some best practices for securing Docker containers?

Answer: Some best practices for securing Docker containers include using official images from trusted sources, minimizing the attack surface by removing unnecessary components, updating images and containers regularly, implementing least privilege principles, using network segmentation, and monitoring container activity.

22. Docker Swarm:

Question 64: How does Docker Swarm manage container scheduling?

Answer: Docker Swarm uses a decentralized and distributed scheduling algorithm to manage container scheduling across the cluster. Swarm managers assign tasks to

worker nodes based on resource availability, service constraints, placement preferences, and other factors.

Question 65: What are the key components of a Docker Swarm cluster?

Answer: The key components of a Docker Swarm cluster include Swarm managers, which orchestrate and manage the cluster, and Swarm nodes, which execute containerized tasks. Swarm managers maintain the cluster state, handle scheduling, and manage worker nodes, while Swarm nodes run containers and report their status to managers.

Question 66: How can you join a Docker Swarm as a worker node?

Answer: You can join a Docker Swarm as a worker node by running the `docker swarm join` command on a host and providing the Swarm manager's address along with the join token generated by the manager. This command joins the host to the Swarm cluster as a worker node.

23. Docker Monitoring and Logging:

Question 67: What are some popular tools for monitoring Docker containers?

Answer: Some popular tools for monitoring Docker containers include Prometheus, cAdvisor, Docker Stats, Datadog, New Relic, and Sysdig. These tools provide insights into container performance, resource usage, and health metrics.

Question 68: How can you collect and centralize Docker container logs?

Answer: You can collect and centralize Docker container logs by configuring logging drivers in Docker daemon settings or individual container configurations. Logging drivers can redirect container logs to external logging systems like Elasticsearch, Splunk, Fluentd, or centralized log management solutions.

Question 69: What is cAdvisor, and how does it monitor Docker containers?

Answer: cAdvisor (Container Advisor) is an open-source container monitoring tool developed by Google. It collects and exports metrics about running containers, including CPU usage, memory usage, filesystem usage, network statistics, and container lifecycle events. cAdvisor runs as a Docker container itself and gathers information from Docker's API.

24. Docker Image Management:

Question 70: What are Docker image layers?

Answer: Docker image layers are the read-only filesystem layers that make up a Docker image. Each layer represents a change to the filesystem, such as adding or modifying files. When you pull or build a Docker image, Docker uses these layers to construct the final image.

Question 71: How can you optimize Docker image size?

Answer: You can optimize Docker image size by using multi-stage builds to reduce the number of layers, minimizing the number of dependencies and unnecessary files in the image, removing temporary files and caches, and using smaller base images like Alpine Linux.

Question 72: What is Docker image caching, and how does it work?

Answer: Docker image caching is a mechanism used during the build process to speed up subsequent builds by reusing intermediate layers from previous builds. When a Dockerfile is built, each instruction produces a layer, and Docker caches these layers to avoid rebuilding them if the instruction and its context haven't changed.

25. Docker Registry:

Question 73: What is Docker Registry?

Answer: Docker Registry is a service for storing and distributing Docker images. It can be either Docker Hub (public registry) or a private registry like Docker Trusted Registry (DTR) or a self-hosted registry using Docker Registry open-source software.

Question 74: How can you push a Docker image to a private registry?

Answer: You can push a Docker image to a private registry using the `docker push` command, followed by the image name and tag, along with the URL of the private registry and authentication credentials if required.
Question 75: What are some security considerations when using Docker Registry?

Answer: Some security considerations when using Docker Registry include enabling access control and authentication, using HTTPS for secure communication, enabling image signing and verification with Docker Content Trust, scanning images for vulnerabilities, and monitoring registry activity.

26. Docker in CI/CD Pipeline:

Question 76: How can you integrate Docker into a CI/CD pipeline?

Answer: Docker can be integrated into a CI/CD pipeline by using Docker images as build environments, running tests inside Docker containers, building Docker images as part of the pipeline, and deploying applications using Docker containers to various environments.

Question 77: What are some benefits of using Docker in a CI/CD pipeline?

Answer: Some benefits of using Docker in a CI/CD pipeline include consistent build environments across different stages of the pipeline, reproducible builds, faster build times with image caching, simplified dependency management, and easier deployment and scaling of applications.

Question 78: What are some CI/CD tools that support Docker integration?

Answer: Some CI/CD tools that support Docker integration include Jenkins, GitLab CI/CD, Travis CI, CircleCI, TeamCity, and GitHub Actions. These tools provide native support for Docker, allowing you to build, test, and deploy applications using Docker containers.

27. Docker for Microservices:

Question 79: How can Docker facilitate microservices architecture?

Answer: Docker facilitates microservices architecture by providing lightweight and isolated containers for individual microservices, enabling seamless deployment, scaling, and management of microservices, allowing each microservice to be developed and tested independently, and simplifying service discovery and communication.

Question 80: What are some challenges of deploying microservices with Docker?

Answer: Some challenges of deploying microservices with Docker include managing container orchestration complexity, ensuring service discovery and communication between microservices, maintaining consistent environments across development, testing, and production, and managing container networking and security at scale.

Question 81: How can you deploy a multi-container microservices application with Docker Compose?

Answer: You can deploy a multi-container microservices application with Docker Compose by defining each microservice as a separate service in a Docker Compose

file, specifying dependencies, networks, volumes, and environment variables, and then using `docker-compose up` to launch the entire application stack.

28. Docker Orchestration:

Question 82: What is Docker orchestration?

Answer: Docker orchestration is the process of managing and coordinating the deployment, scaling, and operation of Docker containers across a cluster of hosts. It involves tasks such as load balancing, service discovery, scheduling, and health monitoring.

Question 83: What are some Docker orchestration tools?

Answer: Some popular Docker orchestration tools include Docker Swarm (built-in), Kubernetes, Apache Mesos, Amazon ECS (Elastic Container Service), and Google Kubernetes Engine (GKE). These tools provide features for automating container management and scaling.

Question 84: Explain the difference between Docker Swarm and Kubernetes.

Answer: Docker Swarm is a simple and easy-to-use container orchestration tool provided by Docker, while Kubernetes is a more feature-rich and complex orchestration platform originally developed by Google. Kubernetes offers advanced scheduling, service discovery, and scaling capabilities compared to Docker Swarm.

29. Docker High Availability:

Question 85: How can you achieve high availability with Docker Swarm?

Answer: High availability with Docker Swarm can be achieved by running multiple Swarm manager nodes for redundancy, enabling automatic service rescheduling, using load balancing for distributing traffic, and implementing health checks to detect and recover from container failures.

Question 86: What is Docker service scaling?

Answer: Docker service scaling refers to the ability to increase or decrease the number of replicas of a service running in a Docker Swarm cluster. Scaling allows you to distribute workload across multiple containers to handle varying levels of traffic and ensure high availability.

Question 87: How does Docker handle node failures in a Swarm cluster?

Answer: Docker Swarm handles node failures by automatically rescheduling failed containers on healthy nodes, based on the desired state defined in the service configuration. Swarm managers monitor the health of nodes and containers, and if a node becomes unavailable, Swarm reschedules the affected containers to maintain service availability.

30. Docker Monitoring and Logging:

Question 88: What are some popular tools for monitoring Docker containers?

Answer: Some popular tools for monitoring Docker containers include Prometheus, cAdvisor, Docker Stats, Datadog, New Relic, and Sysdig. These tools provide insights into container performance, resource usage, and health metrics.

Question 89: How can you collect and centralize Docker container logs?

Answer: You can collect and centralize Docker container logs by configuring logging drivers in Docker daemon settings or individual container configurations. Logging drivers can redirect container logs to external logging systems like Elasticsearch, Splunk, Fluentd, or centralized log management solutions.

31. Docker Swarm:

Question 90: What is Docker Swarm?

Answer: Docker Swarm is a clustering and orchestration tool provided by Docker for managing a cluster of Docker hosts. It enables you to deploy, scale, and manage containers across multiple hosts, providing high availability and load balancing for containerized applications.

Question 91: How do you initialize a Docker Swarm?

Answer: You can initialize a Docker Swarm by running the command `docker swarm init` on a Docker host, which initializes the host as a Swarm manager and generates a join token that other hosts can use to join the Swarm.

Question 92: Explain the difference between Docker Swarm mode and Docker standalone mode.

Answer: Docker Swarm mode is a built-in feature of Docker Engine that provides clustering and orchestration capabilities, allowing you to create and manage a Swarm cluster for deploying and scaling containers. Docker standalone mode refers to running Docker without clustering or orchestration features.

32. Docker Security:

Question 93: What is Docker Content Trust (DCT)?

Answer: Docker Content Trust is a security feature that allows you to verify the authenticity and integrity of Docker images. When enabled, DCT uses cryptographic signatures to ensure that only trusted images are pulled and run on Docker hosts.

Question 94: How can you improve Docker container security?

Answer: You can improve Docker container security by using official images from trusted sources, regularly updating images and containers, minimizing the attack surface by using the principle of least privilege, implementing network segmentation, using Docker Content Trust to verify image integrity, and monitoring container activity.

Question 95: What are Docker security best practices?

Answer: Docker security best practices include using least privilege, keeping images and containers up to date, avoiding running containers as root, using Docker Content Trust, implementing network segmentation, scanning images for vulnerabilities, and monitoring container activity.

33. Docker Monitoring and Logging:

Question 96: How can you monitor Docker containers?

Answer: You can monitor Docker containers using tools like Docker Stats, Docker Events, cAdvisor, Prometheus with Grafana, and third-party monitoring solutions. These tools provide insights into container resource usage, performance metrics, and health status.

Question 97: What are some logging options available in Docker?

Answer: Docker provides logging drivers that allow you to control how container logs are handled. Some logging options include the `json-file` driver (default), `syslog`, `journald`, `gelf`, `fluentd`, `awslogs`, and `splunk`.
Question 98: How can you view container logs in Docker?

Answer: You can view container logs in Docker using the `docker logs` command followed by the container ID or name. Additionally, you can use logging drivers to redirect container logs to external logging systems for centralized log management.

34. Docker Networking:

Question 99: What is Docker bridge network?

Answer: Docker bridge network is the default network mode in Docker, which creates a virtual bridge on the host system and assigns IP addresses to containers connected to that bridge. Containers on the same bridge network can communicate with each other using IP addresses, and the bridge network provides NAT (Network Address Translation) for external connectivity.

Question 100: How can you create a custom bridge network in Docker?

Answer: You can create a custom bridge network using the `docker network create` command, specifying the `--driver bridge` option followed by the desired network name.

35. Docker Compose:

Question 101: What is Docker Compose?

Answer: Docker Compose is a tool for defining and running multi-container Docker applications. It uses YAML files to configure the application's services, networks, and volumes, making it easy to manage complex Docker setups.

Question 102: How do you define services in a Docker Compose file?

Answer: Services in a Docker Compose file are defined under the `services` section using YAML syntax, where each service specifies its image, ports, environment variables, volumes, and other configuration options.