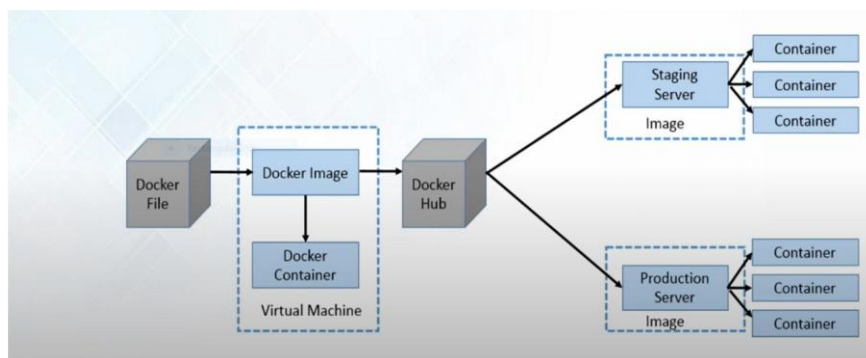




# Docker Interview Questions

## Can you describe a situation where you used Docker to solve a specific problem?

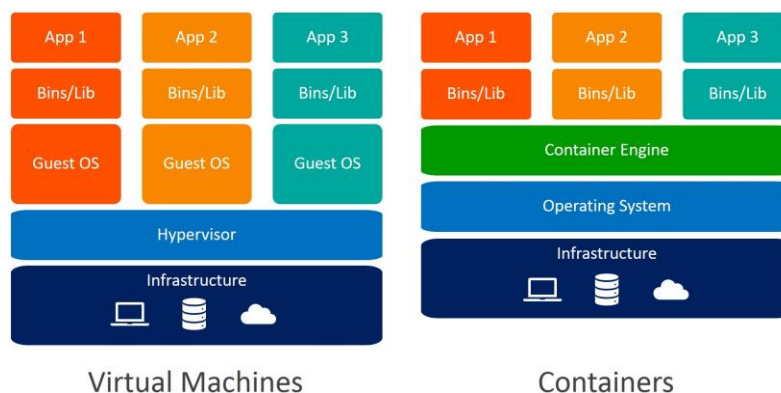
Yes. In a recent project, we faced inconsistency issues between development and production environments, leading to deployment errors. To solve this, we adopted Docker. And deployed the application with Kubernetes



## Can you explain how Docker container differs from virtual machines?

Docker containers share the host OS kernel, making them lightweight and efficient. They provide process-level isolation and package applications with their dependencies.

Virtual machines, in contrast, run full guest OSs on a hypervisor, consuming more resources and offering stronger isolation.



**You're in charge of maintaining Docker environments in your company and You've noticed many stopped containers and unused networks taking up space.**

**Describe how you would clean up these resources effectively to optimize the Docker environment.**

The `docker prune` command is used to clean up unused Docker resources, such as containers, volumes, networks, and images. It helps reclaim disk space and tidy up the Docker environment by removing objects that are not in use.

There are different types of `docker prune` commands:

- `docker container prune` : Removes stopped containers.
- `docker volume prune` : Deletes unused volumes.
- `docker network prune` : Cleans up unused networks.
- `docker image prune` : Removes unused images.

Running `docker system prune` combines these functionalities into one command, ensuring that Docker removes any resources not associated with a running container.

**Note:** The `docker system prune` command should be used with caution as it permanently deletes unused resources. Any data associated with them will be lost.

```
~$ docker system df
TYPE                TOTAL        ACTIVE        SIZE        RECLAIMABLE
Images              12           12           12.79GB     1.838GB (14%)
Containers          19           19           31.25MB     0B (0%)
Local Volumes       0            0            0B          0B
Build Cache         0            0            0B          0B

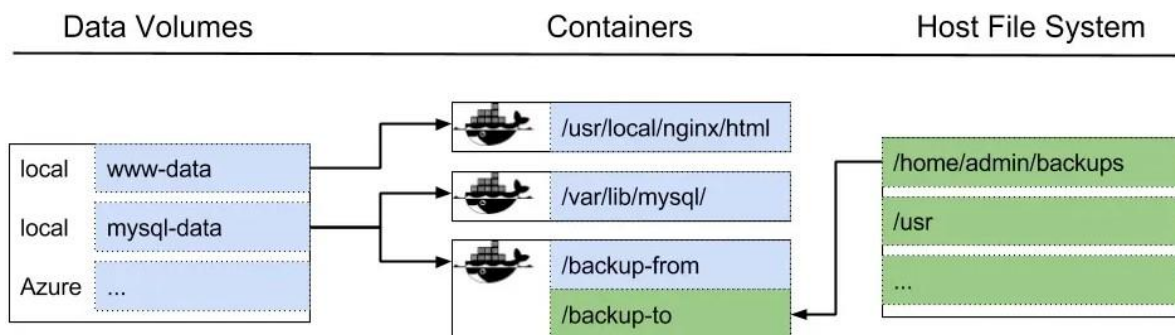
~$ docker system prune -a
WARNING! This will remove:
- all stopped containers
- all networks not used by at least one container
- all images without at least one container associated to them
- all build cache

Are you sure you want to continue? [y/N] y
Total reclaimed space: 0B
```

## You're working on a project that requires Docker containers to persistently store data

### How do you handle persistent storage in Docker?

Docker volumes can be used for persistent storage. They are managed by Docker and can be attached to one or more containers. Another approach is to use bind mounts, which are linked to the host file system.



Learning Resource:

<https://docs.docker.com/guides/walkthroughs/persist-data/>

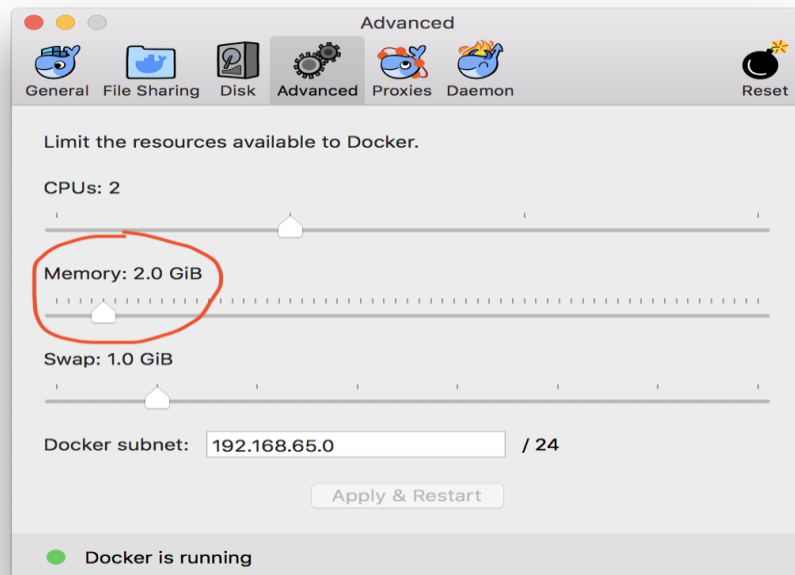
## A company wants to create thousands of Containers. Is there a limit on how many containers you can run in Docker?

The amount of containers that may be run under Docker has no explicitly specified limit. But it all relies on the constraints, particularly the hardware constraints. The size of the program and the number of CPU resources available are two major determinants of this restriction. If your program isn't too large and you have plenty of CPU resources, we can run a lot of containers.

**You're managing a Docker environment and need to ensure that each container operates within defined CPU and memory limits.  
How do you limit the CPU and memory usage of a Docker container?**

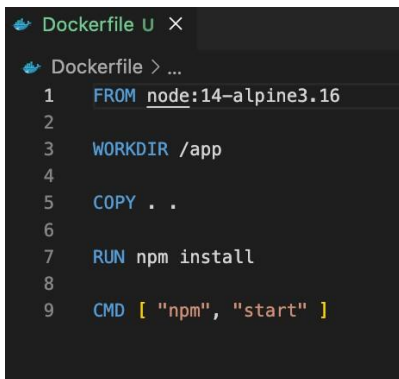
Docker allows you to limit the CPU and memory usage of a container using resource constraints. You can set the CPU limit with the `--cpu` option and the memory limit with the `--memory` option when running the container using the `docker run` command.

For example, `docker run --cpu 2 --memory 1g mycontainer` limits the container to use a maximum of CPU cores and 1GB of memory.



## What is Dockerfile and how is it used in Docker?

A Dockerfile is a text file that contains commands or instructions to build a Docker image. It can specify the base image, the files to add to the container, the commands to run during the build process, and the command to run when the container starts.



```
Dockerfile U X
Dockerfile > ...
1 FROM node:14-alpine3.16
2
3 WORKDIR /app
4
5 COPY . .
6
7 RUN npm install
8
9 CMD [ "npm", "start" ]
```

## What is the purpose of the CMD instruction in a Dockerfile?

The `CMD` instruction in a Dockerfile specifies the default command to be executed when a container is started from the Docker image.

It defines the primary functionality of the container, such as running an application or executing a script. If a Docker container is started without specifying a command, the command specified in the `CMD` instruction will be executed by default.

**You've been tasked with ensuring the application can handle increased loads by scaling Docker containers horizontally.**

### **How do you scale Docker containers horizontally?**

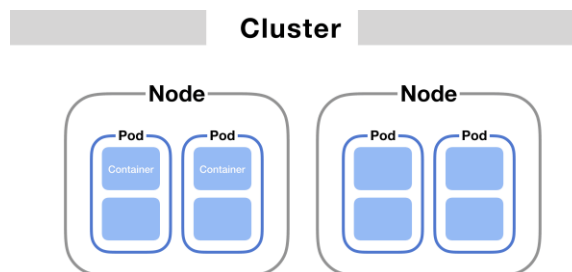
To scale Docker containers horizontally, you can use Docker Swarm or a container orchestration tool like Kubernetes.

Which lets you create and manage multiple docker containers defining the number of replicas in declarative way. (Manifests)

### **What is the difference between a Docker container and a Kubernetes pod?**

A Docker container is a lightweight and isolated runtime environment that runs a single instance of an application. It is managed by Docker and provides process-level isolation.

On the other hand, a Kubernetes pod is a higher-level abstraction that can contain one or more Docker containers (or other container runtimes).



**You're part of a development team deploying a microservices architecture using Docker containers. One of the containers, critical to the system's functionality, has suddenly started failing without clear error messages. How do you debug issues in a failing Docker container?**

There are several techniques to debug issues in a Docker container:

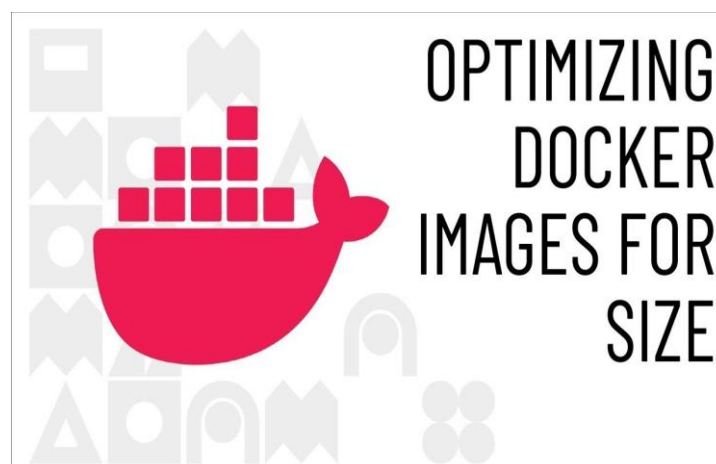
**Logging:** Docker captures the standard output and standard error streams of containers, making it easy to inspect logs using the `docker logs` command.

**Shell access:** You can access a running container's shell using the `docker exec` command with the `-it` option. This allows you to investigate and troubleshoot issues interactively.

**Image inspection:** You can inspect the Docker image's contents and configuration using `docker image inspect`. This lets you check for potential misconfigurations or missing dependencies.

**Health checks:** Docker supports defining health checks for containers, allowing you to monitor the health status and automatically restart or take action based on predefined conditions.

**Can you describe a situation where you optimized a Dockerfile for faster build times or smaller image size?**



Optimizing a Dockerfile could involve various strategies like using a smaller base image, reducing the number of layers by combining commands, or using multi-stage builds to exclude unnecessary files from the final image

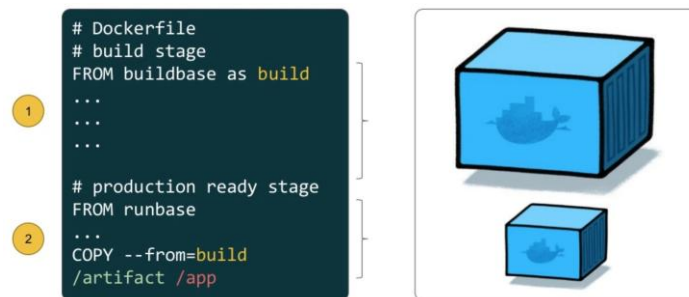
## How do you create a multi-stage build in Docker?

Multi-stage builds in Docker allow you to create optimized Docker images by leveraging multiple build stages.

To create a multi-stage build, you define multiple **FROM** instructions in the Dockerfile, each representing a different build stage.

Each stage can use a different base image and perform specific build steps.

Intermediate build artifacts can be copied between stages using the **COPY --from** instruction. This technique helps reduce the image size by excluding build tools and dependencies from the final image.



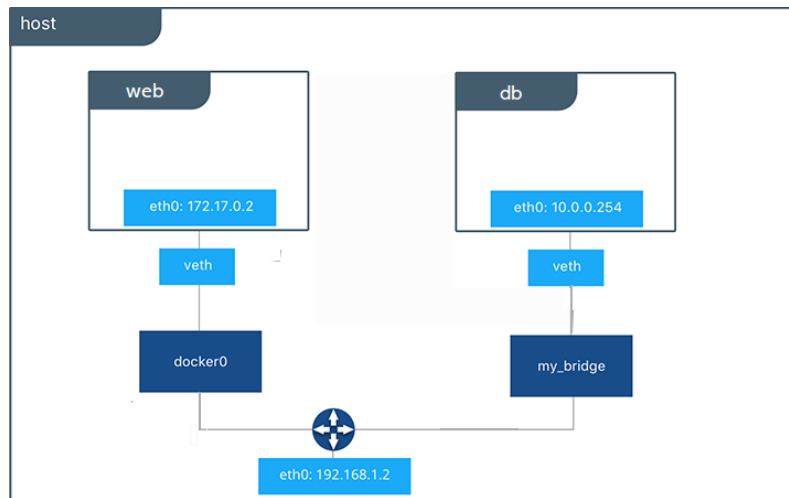
## How do you create a custom Docker network?

To create a custom Docker network, you can use the **docker network create** command followed by the desired network name. For example:

```
docker network create mynetwork
```

This command creates a new custom network named "mynetwork" using the default bridge driver. You can also specify a different driver using the **-driver** option if needed.





**You're working on a critical application running in Docker containers, and an update needs to be applied without risking data loss. How do you update a Docker container without losing data?**

The steps to update a Docker container without losing data are:

- Create a backup of any important data stored within the container.
- Stop the container gracefully using the `docker stop` command.
- Pull the latest version of the container image using `docker pull`.
- Start a new container using the updated image, making sure to map any necessary volumes or bind mounts.
- Verify that the new container is functioning correctly and that the data is still intact.

**You're responsible for securing the Docker containers hosting your organization's sensitive applications.  
How do you secure Docker containers?**

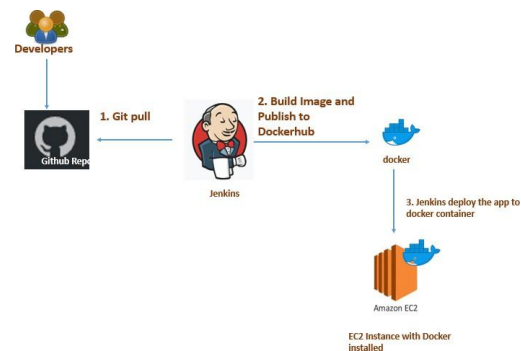
Securing Docker containers involves implementing a multi-layered approach. Some recommended practices include:

- Using trusted base images from reputable sources.
- Regularly updating Docker and the underlying host system with security patches.
- Scanning container images for vulnerabilities using security tools like docker scout.
- Implementing network segmentation and access controls.
- Monitoring container activity and logging for security analysis.
- Applying the least privilege principle and implementing container-specific security measures like AppArmor or seccomp profiles.



## Have you ever used Docker in combination with a continuous integration/continuous deployment (CI/CD) system?

Yes, in nearly all my projects. For example, in one project, we integrated Docker with Jenkins. In the build stage, Jenkins would trigger a job that builds a new Docker image and pushes it to a Docker registry. During the deployment stage, this image was pulled from the registry and deployed to the production environment using Kubernetes. This approach ensured uniformity across all stages of the deployment process.



## How is Docker used within AWS?

In my previous project, we utilized Docker within AWS using Amazon's Elastic Container Service (ECS). We deployed our microservices as Docker containers on ECS, which enabled us to efficiently manage the lifecycle of these services. We also used AWS Fargate for serverless compute for our containers.



## How do you monitor Docker containers?

There are various ways to monitor Docker containers, including:

- Using Docker's built-in container monitoring commands, such as **docker stats** and **docker container stats**, to view resource usage statistics.
- Integrating with container monitoring and logging tools like Prometheus, Grafana, or ELK stack (Elasticsearch, Logstash, Kibana) to collect and analyze container metrics and logs.
- Leveraging container orchestration platforms that offer built-in monitoring capabilities such as Docker Swarm's service metrics or Kubernetes' metrics API.
- Using specialized monitoring agents or tools that provide container-level insights and integration with broader monitoring and alerting systems.

## What are some best practices for using Docker in production environments?

Some best practices for using Docker in production environments include:

- Building and using lightweight container images to improve deployment speed and reduce the attack surface.
- Regularly updating Docker and the underlying host system with security patches.
- Implementing container orchestration platforms, such as Docker Swarm or Kubernetes, to manage containers at scale and provide features like load balancing and service discovery.
- Configuring resource limits for containers to prevent resource contention and ensure fair allocation.
- Monitoring container health, resource usage, and application metrics for performance optimization and troubleshooting.
- Implementing secure network configurations such as using private networks and encrypting container traffic.
- Backing up critical data and using volume or storage solutions that provide data persistence and redundancy.
- Implementing a comprehensive security strategy, including container vulnerability scanning, access controls, and least privilege principles.

## How do you use Docker with Kubernetes?

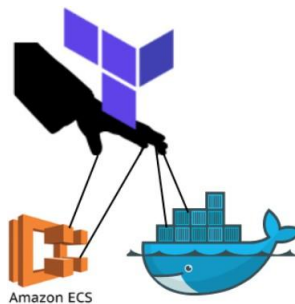
Docker can be used as the container runtime within a Kubernetes cluster. When deploying applications on Kubernetes, Docker is responsible for creating and managing containers on each node.

Kubernetes uses the Docker API to interact with Docker and perform container-related operations such as pulling images, creating containers, and managing their lifecycle. Docker images are typically stored in a container registry accessible to the Kubernetes cluster.

## Can you describe a situation where you used Terraform to automate the deployment of Docker containers?

In a recent project, we used Terraform to automate the deployment of our Docker containers on AWS. We defined our infrastructure as code using Terraform, which included our ECS services and tasks.

These tasks specified the Docker images to use, pulled from ECR, and the resources allocated to each container. Whenever we made changes to our Terraform scripts, we could easily apply these changes to update our infrastructure.



## How do you implement canary deployments with Docker and Kubernetes?

Canary deployments with Docker and Kubernetes can be implemented by following these steps:

- Set up a Kubernetes cluster and deploy your application using Docker containers.
- Create a new deployment or service for the updated version of your application (the Canary version).
- Configure the canary deployment to receive a small percentage of traffic while the majority of traffic is still routed to the stable version.
- Monitor the canary deployment, collecting metrics and user feedback to assess its stability and performance.
- If the canary deployment proves successful, gradually increase its traffic allocation.
- If any issues arise, quickly roll back by reducing the canary's traffic allocation or rolling back the deployment.
- Repeat the process, gradually increasing the canary's traffic allocation until it becomes the new stable version.

### Microservices – Canary Deployments

