**Assignment**

* What is Docker, and why is it used?

Docker is a containerization platform that allows deveopers to package ,ship,, and run applications in containers

USES

1. Conatinerzation
2. Lightweight
3. Portability
4. Isolation
5. Security
6. Scalability
7. Easy anf fast deployment
8. Large community
9. Resource efficiency

* How is Docker different from a virtual machine (VM)?

Docker and virtual machines (VMs) are both used for virtualization, but they differ in their approach and functionality:

1. **Virtualization Technology: VMs** use hypervisor-based virtualization, which creates a complete, self-contained operating environment. **Docker** uses containerization, which relies on the host operating system's kernel.
2. **Resource Usage: VMs** require a dedicated allocation of system resources (CPU, memory, and storage), whereas **Docker** containers share the host's resources.
3. **Operating System:** Each VM runs its own operating system, whereas Docker containers share the host's operating system.
4. **Isolation: VMs** provide a higher level of isolation between virtual machines, whereas **Docker** containers provide a lower level of isolation, relying on kernel namespaces and control groups
5. **Portability**: **Docker** containers are highly portable, as they can run on any system that supports Docker, without requiring specific dependencies or configurations. **VMs** are less portable, as they require a specific hypervisor and may have compatibility issues.
6. **Performance:** Docker containers have faster startup times and lower overhead compared to VMs.
7. **Security: Docker** containers provide a secure way to deploy applications, as they can define access controls and network policies. **VMs** also provide security, but may require additional configuration.

* What are the main components of Docker?

**1. Docker Engine**

The Docker Engine is the core component of Docker. It is a lightweight runtime environment that manages containers on a host system.

**2. Docker Hub**

Docker Hub is a cloud-based registry that stores and manages Docker images. It provides a centralized location for users to find, share, and manage Docker images.

**3. Docker Images**

Docker images are templates that contain the application code, dependencies, and configurations. They are used to create containers.

**4. Docker Containers**

Docker containers are runtime instances of Docker images. They provide a isolated environment for applications to run in.

**5. Docker Volumes**

Docker volumes provide a way to persist data between container restarts. They allow you to store data outside of the container's file system.

**6. Docker Networks**

Docker networks provide a way to connect multiple containers together. They allow you to create a network topology that enables communication between containers.

**7. Docker Compose**

Docker Compose is a tool for defining and running multi-container Docker applications. It provides a way to define the services, networks, and volumes required by an application.

**8. Docker Swarm**

Docker Swarm is a container orchestration tool that allows you to manage multiple containers across multiple hosts. It provides a way to deploy, manage, and scale containerized applications.

**9. Dockerfile**

A Dockerfile is a text file that contains instructions for building a Docker image. It provides a way to automate the process of creating a Docker image.

**10. Docker CLI**

The Docker CLI (Command Line Interface) is a command-line tool that allows you to interact with the Docker Engine. It provides a way to manage containers, images, volumes, and networks.

* Explain the difference between Docker images and Docker containers.

**Docker Images**

1. **Template:** A Docker image is a template that contains the application code, dependencies, and configurations.
2. **Immutable:** Docker images are immutable, meaning they cannot be changed once they are created.
3. **Portable:** Docker images are portable and can be run on any system that supports Docker.
4. **Stored in Docker Hub:** Docker images can be stored in Docker Hub, a cloud-based registry.

**Docker Containers**

1. **Runtime instance:** A Docker container is a runtime instance of a Docker image.
2. **Mutable:** Docker containers are mutable, meaning they can be changed and updated while they are running.
3. **Isolated:** Docker containers are isolated from each other and from the host system.
4. **Ephemeral:** Docker containers are ephemeral, meaning they can be created and deleted as needed.

**Differences**

1. **Immutability:** Docker images are immutable, while Docker containers are mutable.
2. **Portability:** Docker images are portable, while Docker containers are not.
3. **Isolation:** Docker containers provide isolation, while Docker images do not.
4. **Ephemerality:** Docker containers are ephemeral, while Docker images are not.

* What is a Dockerfile?

A Dockerfile is a text file that contains instructions for building a Docker image. It is a recipe for creating a Docker image, specifying the base image, copying files, installing dependencies, and setting environment variables.

**Components of a Dockerfile**

1. **FROM:** Specifies the base image for the new image
2. **RUN:** Executes a command during the build process.
3. **COPY:** Copies files from the host machine to the container.
4. **ADD:** Copies files from the host machine to the container, with additional functionality for handling tar archives and remote URLs.
5. **ENV:** Sets environment variables for the container.
6. **EXPOSE:** Exposes a port from the container to the host machine.
7. **CMD:** Specifies the default command to run when the container starts.
8. **ENTRYPOINT:** Specifies the entry point for the container.
9. **VOLUME:** Creates a volume in the container.
10. **WORKDIR:** Sets the working directory in the container.

**Benefits of Using a Dockerfile**

1. **Automated build process:** A Dockerfile automates the build process, ensuring consistency and reproducibility.
2. **Version control:** A Dockerfile can be version-controlled, allowing for easy tracking of changes.
3. **Sharing and collaboration:** A Dockerfile can be shared and used by others, facilitating collaboration and knowledge sharing.

* What command is used to build a Docker image?

**Docker Build Command:**

docker build [options] PATH

**Options**

- -t: Specify the name and tag for the image.

- -f: Specify the Dockerfile to use.

- --no-cache: Disable the cache for the build process.

**Example**

docker build -t my-image .

* How do you run a container from an image?

**Docker Run Command**

docker run [options] IMAGE [COMMAND]

Options

- -i: Keep the container running and attach to it.

- -t: Allocate a pseudo-TTY to the container.

- -d: Run the container in detached mode.

- -p: Publish a container's port to the host machine.

- --name: Specify a name for the container.

**Example**

docker run -it -p 8080:8080 my-image

* How do you list all running containers?

**Docker Ps Command**

docker ps

**Options**

- -a: Show all containers, including stopped ones.

- -q: Only display the container IDs.

- --filter: Filter the output based on conditions such as container name, status, or volume.

**Example**

docker ps -a

* What command is used to stop a running container?

**Docker Stop Command**

docker stop CONTAINER\_ID

**Options**

- -t: Specify the timeout before forcing the container to stop.

**Example**

docker stop my-container

* How do you remove a Docker container?

**Docker Rm Command**

docker rm CONTAINER\_ID

**Options**

- -f: Force the removal of the container, even if it's running.

- -v: Remove the volumes associated with the container.

**Example**

docker rm my-container

This command removes the container with the ID or name "my-container".

**Removing Multiple Containers:** You can remove multiple containers at once by specifying their IDs or names

docker rm container1 container2 container3

**Removing All Stopped Containers:** You can remove all stopped containers using the following command:

docker container prune

* What is a Docker volume, and why is it used?

Docker volumes are directories or files that are shared between a container and the host machine. They provide a way to persist data even after a container is deleted or restarted.

**Uses of Docker Volumes:**

1. **Persisting Data:** Docker volumes can be used to persist data even after a container is deleted or restarted.
2. **Sharing Data:** Docker volumes can be used to share data between multiple containers.
3. Configuring Containers: Docker volumes can be used to configure containers by mounting configuration files or directories.
4. **Logging:** Docker volumes can be used to store log files from containers

**Types of Docker Volumes:**

1. **Named Volumes:** Named volumes are volumes that are given a specific name. They can be used to persist data and can be shared between multiple containers.
2. **Anonymous Volumes:** Anonymous volumes are volumes that are not given a specific name. They are used to persist data for a single container.
3. **Bind Mounts:** Bind mounts are volumes that are mounted from the host machine to a container.

**Creating a Docker Volume:**

1. **Using the -v Flag:** You can create a Docker volume using the -v flag when running a container. For example:  
   docker run -v my-volume:/app/data my-image
2. **Using the Docker Volume Command:** You can also create a Docker volume using the Docker volume command. For example:

docker volume create my-volume

**Managing Docker Volumes:**

1. **Listing Volumes:** You can list all Docker volumes using the following command:  
   docker volume ls
2. **Inspecting Volumes:** You can inspect a Docker volume using the following command:  
   docker volume inspect my-volume
3. **Removing Volumes:** You can remove a Docker volume using the following command:

docker volume rm my-volume

* What is the difference between CMD and ENTRYPOINT in a Dockerfile?
* How do you persist data in Docker containers?
* What is a Docker Compose file? How is it used?

A Docker Compose file is a YAML file that defines and configures multiple Docker containers and services. It provides a way to define and manage complex Docker applications with multiple services.

**Benefits of Using Docker Compose:**

1. Simplified service management: Manage multiple services with a single command.
2. Improved service orchestration: Define dependencies and scaling for your services.
3. Easier development and testing: Quickly spin up and down complex applications for development and testing.
4. Environment variable management: Define environment variables for each service.
5. Volume management: Mount volumes to persist data

* How do you scale services using Docker Compose?
* How do you check the logs of a running container?

**Docker Logs Command:**

docker logs [OPTIONS] CONTAINER\_ID

**Options:**

- -f: Follow the logs, showing new logs as they are generated.

- --since: Show logs since a specific time.

- --until: Show logs until a specific time.

- -n: Show the last n lines of logs.

- --tail: Show the last n lines of logs.

**Example:**

docker logs -f my-container

* What is the purpose of the .dockerignore file?

The .dockerignore file is a text file that is used to specify files and directories that should be excluded from the Docker build context.

**Purpose of .dockerignore:**

1. **Exclude unnecessary files**: The .dockerignore file helps to exclude unnecessary files and directories from the Docker build context, which can speed up the build process.
2. **Prevent sensitive data exposure:** By excluding sensitive files and directories, such as those containing passwords or API keys, you can prevent them from being exposed in the Docker image.
3. **Reduce Docker image size:** By excluding unnecessary files and directories, you can reduce the size of the Docker image.

* What are the different networking modes in Docker?

**Docker Networking Modes:**

1. **Bridge Mode:** This is the default networking mode in Docker. Containers run in a separate network namespace and communicate with each other through a virtual bridge.
2. **Host Mode:** In this mode, containers use the host machine's network stack, allowing them to communicate directly with the host machine and other containers on the same host.
3. **None Mode:** This mode disables networking for a container, isolating it from the host machine and other containers.
4. **Container Mode:** This mode allows a container to join the network namespace of another container.
5. **Macvlan Mode:** This mode allows containers to be connected to a physical network interface, providing a direct connection to the physical network.
6. **Overlay Mode:** This mode allows containers to communicate with each other across multiple hosts, using a virtual network that spans the hosts.

* How do you expose ports in a Docker container?

1. **Docker Run Command:**  
   You can expose ports using the -p flag when running a container.  
   docker run -p host\_port:container\_port my\_image  
   Example:  
   docker run -p 8080:80 my\_web\_server  
   This command maps port 8080 on the host machine to port 80 in the container.
2. **Docker Compose:**  
   You can expose ports in a Docker Compose file using the ports directive.  
   version: '3'  
   services:  
    web:  
    image: my\_web\_server  
    ports:  
    - "8080:80"
3. **Dockerfile:**  
   You can expose ports in a Dockerfile using the EXPOSE instruction.  
   FROM my\_base\_image  
   EXPOSE 80

This instruction exposes port 80 in the container.

* What is the difference between docker stop and docker kill?

Docker stop and docker kill are two commands used to terminate a running Docker container. However, they differ in their approach and behavior:

**Docker Stop:**

1. **Sends a SIGTERM signal:** docker stop sends a SIGTERM signal to the container's process, allowing it to shut down gracefully.

**2. Waits for the container to exit:** Docker waits for the container to exit before considering the stop operation complete.

3**. Default timeout:** If the container doesn't exit within 10 seconds (default timeout), Docker sends a SIGKILL signal to force the container to exit.

**Docker Kill:**

1**. Sends a SIGKILL signal:** docker kill sends a SIGKILL signal to the container's process, forcing it to exit immediately.

2. **Does not wait for the container to exit:** Docker does not wait for the container to exit before considering the kill operation complete.

3**. No timeout:** There is no timeout for the docker kill command; the container is forced to exit immediately.

**Differences:**

1. **Graceful shutdown**: docker stop allows for a graceful shutdown, while docker kill forces an immediate exit.

2**. Timeout:** docker stop has a default timeout, while docker kill does not.

3. **Signal sent:** docker stop sends a SIGTERM signal, while docker kill sends a SIGKILL signal.

**When to use each:**

1. Use docker stop when you want to allow the container to shut down gracefully, such as during maintenance or when updating a service.

2. Use docker kill when you need to force a container to exit immediately, such as in emergency situations or when a container is not responding.