**Assignment**

1. **What is Docker, and why is it used?**

* Docker is a containerization software.
* Docker is a virtualization software.
* It is one of the Devops tool.
* It is written in ‘GO’ language.
* It was released in 2013.
* Developed by Solomon Hykes and Sebastian phal.
* Makes developing and deploying applications much easier.
* Docker is a set of platforms as a service product that use operating system level virtualization to deliver software in packages called containers.
* Docker is a free and open-source platform that automates the deployment, scaling, and management of applications with lightweight, portable containers.
* **Why docker:**
* Docker is an open platform for developing, shipping, and running applications.
* Using docker lets you ship code faster, standardized application operations, seamlessly move code, and save money by improving resource utilization.
* **Consistency across environments**: Docker containers encapsulate everything an application needs to run, ensuring it behaves the same regardless of where it is deployed (development, testing, production).
* **Isolation**: Each container runs in its own environment, which means applications do not interfere with each other. This isolation helps manage dependencies and reduce conflicts.
* **Portability**: Docker container runs on any system that supports docker.
* **Scalability**: Docker makes it simple to scale applications up or down by adding or removing containers as needed. This is particularly useful for handling varying loads.
* **Simplified CI/CD**: Docker integrates seamlessly with CI/CD pipelines, facilitating automated testing, building, and deployment processes, which accelerates software delivery.
* **Version control**: Docker images can be versioned, making it easy to roll back to previous versions of an application or to deploy specific versions consistently.
* **Community and ecosystem**: Docker has a large community and a rich ecosystem of tools and resources, including docker hub for sharing images and various orchestration tools like Kubernetes.

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

* Virtual machines run as virtual environments on the same hardware.
* Docker runs on virtualizations of the same operating system.

|  |  |  |
| --- | --- | --- |
| Feature | Virtual machine | Docker |
| Virtualization layer | Virtual machines use a hypervisor to virtualize the hardware. | Docker uses the host operating systems kernel to virtualize the environment. |
| Operating system | Virtual machines require a separate operating system for each virtual machine. | Docker containers share the host operating system. |
| Resource usage | Virtual machines require more resources (CPU, memory, and storage) than docker containers. | Docker containers, since each VM has its own operating system and virtualized hardware. |
| Portability | Compare with docker container less portable. | More portable than Virtual machines. |
| Isolation | Virtual machines provide a higher level of isolation. Since each VM has its own operating system and virtualized hardware. | Docker provides a higher level of isolation. |
| Startup time | As compare with docker container virtual machine is slow. | Docker containers start much faster than VMs. |
| Size | Virtual machines are large. | Docker containers are typically much smaller than virtual machines, since they don't require a separate operating system. |

3. **What are the main components of Docker?**

1. **Docker engine**:

* The docker engine is the core component responsible for building, shipping, and running containers.
* It provides a lightweight and portable way to deploy applications.

1. **Docker images**:

* Docker images are read only templates used to create containers.
* They contain the application code, libraries, dependencies, and configurations requires to run an application

1. **Docker containers**:

* A docker container is a runnable instance of a docker image. It includes everything needed to run an application.
* Docker containers are lightweight and portable encapsulations of an application and its dependencies.
* They provide a consistent way to deploy applications.

1. **Docker File**:

* A text file containing a set of instructions for building a docker image.
* It defines the environments and dependencies needed for an application.
* Written by developers.

1. **Docker hub**:

* Docker hub is a cloud-based registry that allows users to store, manage, and share container images.
* It provides a centralized location for discovering and downloading container images.

1. **Docker compose**:

* Docker compose is a tool for defining and running multi container docker applications.
* It provides a way to manage and orchestrate containers.

1. **Docker swarm**:

* A native clustering and orchestration tool for devops.
* Allows users to manage and deploy containerized applications across a cluster of hosts.

1. **Docker volumes**:

* Docker volumes provide persistent storage for containers.
* They allow data to be stored and retrieved even after a container is deleted.

4. **Explain the difference between Docker images and Docker containers.**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Docker image** | **Docker container** |
| Definition | Docker images are read only templates used to create containers. They contain the application code, libraries, dependencies, and configurations requires to run an application. | A docker container is a runnable instance of a docker image. It includes everything needed to run an application. |
| Purpose | Serves as a blueprint for creating containers. | Provides an isolated environment for running applications. |
| Immutability | Immutable (once created it does not change). | Mutable (can be started, stopped, and modified). |
| Creation | Built from a docker file. | Created from a docker image. |
| Storage | Can be stored in a registry. (docker hub). | Runs in memory and is ephemeral. |
| Execution | Not executed directly. | Executed to run applications |
| Size | Generally larger, as it includes all dependencies and files needed to run an application. | Typically, smaller than image, as it only contains the runtime data and changes. |

5**. What is a Docker file?**

* A docker file is a text file that contains a set of instructions for building a docker image.
* It defines the environment and dependencies needed for an application.
* It is written by developers.
* Mutable (can be modified)
* Not stored just a script.
* Not executed directly.
* Docker file name should be capital letter ‘D’.
* File name should be unique.
* Using docker file we can create docker image.

6. **What command is used to build a Docker image?**

* **Docker build**

7. **How do you run a container from an image?**

* To run a container from an image, you can use the docker run command.
* **docker run**
* Docker run [options] image [command] [ARGUMENTS]
* -D run the container in detached mode.
* -i interactive
* -t , --tty: allocate a pseudo- TTY to the container.
* -p, --publish – publish a containers port to the host machine.
* -v, --volume: mount a volume from the host machine to the container.

8. **How do you list all running containers?**

* **docker ps** - This command displays a list of all currently running commands. By default, it only shows active containers.

9. **What command is used to stop a running container?**

* **docker stop <container id or container name>**

10. **How do you remove a Docker container?**

* **docker rm <container id or container name>**
* **docker rm –f <container id or container name>**

11. **What is the difference between CMD and ENTRYPOINT in a Docker file?**

|  |  |  |
| --- | --- | --- |
| **Feature** | **CMD** | **ENTRYPOINT** |
| Purpose | Provides default commands and arguments for the container. | Configures a container to run as an executable. |
| Overriding | Can be overriding by command line arguments when running the container. | Cannot be easily overirder.intended to be the main command. |
| Syntax | Can be specified in two forms:  CMD [“executable”,”para1”,”para2”  CMD command para1 para2. | Typically specified in exec form:  ENTRYPOINT [“executable”,”para1”,”para2”]. |
| Use case | Best used for providing defaults that can be overridden. | Best used for when you want the container to run a specific application or script. |
| Execution environment | Execute in the context of the ENTRYPOINT command if specified. | Executes as the main process of the container. |
| Example | CMD [“nginx”, “-g”, “daemon off;”] | ENTRYPOINT [“nginx”]and CMD [“-g”, “daemon off;”] can be combined. |

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

* A docker volume is a directory that is shared between the host machine and a container. It allows data to be written to the host machines file system, even after the container is deleted.
* **Why use docker volumes**:
* **Data persistence**: volumes ensure that important data, such as databases or user uploads, persists beyond the lifetime of a single container.

It provides a way to persist data even after a container is deleted or restarted.

* **Data sharing**: volumes allow multiple containers to share data easily.

For example: a web application container can share a volume with a database container.

* **Performance**: using volumes can provide better performance compared to storing data in a containers writable layer, especially for large datasets.
* **Ease of backup and** **restore:** volumes can be easily backed up and restored, making it simpler to manage data integrity and recovery.
* **Avoiding data loss:** volumes prevent data loss when a container is deleted or crashes.

13. **How do you persist data in Docker containers?**

* There are several ways to persist data in docker containers:
* **Volumes**: docker volumes provide a way to persist data even after a container is deleted.
* **Bind mounts**: Bind mounts allow you to mount a directory from the host machine to a container.
* **Docker volumes plugin**: docker provides a volumes plugin that allows you to create and manage volumes.
* **Persistent volumes with docker compose**: docker compose provides a way to define persistent volumes for your services.
* **Stateful sets with Kubernetes**: if you are running your containers in a Kubernetes cluster, you can use stateful sets to persist data.

14. **What is a Docker Compose file? How is it used?**

* A docker compose file is a YAML file used to define and configure multi container docker applications.
* It allows developers to specify the services, networks, and volumes required for an application in single file, making it easier to manage and deploy complex applications.
* **Docker compose file structure**:
* Version
* Service
* Networks
* Volumes.
* **Use**:
* Create a file named docker-compose.yml in your project directory and define your services.
* Start the application “docker-compose up -d
* Stop the application docker-compose down
* To view logs docker-compose logs
* Scaling services

For example, to scale the api service to 3 instances “docker-compose up –scale “

15. **How do you scale services using Docker Compose?**

* To scale services using docker compose, you can use the scale command or define the replicas option in the docker-compose.yml file.
* Using the scale command:
* Docker-compose scale

Service\_name=num\_replicas

Example: docker-compose scale web=3.

16. **How do you check the logs of a running container?**

* **docker logs [options] <container name or container id>**

17. **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 ignored by docker when building an image.
* **Purpose of .dockerignore:**
* **Exclude unnecessary files**: ignore files and directories that are not necessary for the docker image, such as source code, build artifacts, or sensitive data.
* **Reduce image size**: by excluding unnecessary files, the resulting docker image will be smaller in size, making it easier to transfer and store.
* **Improve build performance**: ignoring unnecessary files can speed up the docker build process, as there will be fewer files to process.
* Lines starting with # are comments and are ignored
* Lines containing a pattern.\*txt will ignore files matching that pattern
* Lines containing a directory path node\_modules/) will ignore the entire directory.
* Example:
* #ignore source code
* \*.java
* \*.c
* \*.cpp
* #ignore build artifacts
* Target/
* Build/
* #ignore sensitive data
* Secrets.properties.

18. **What are the different networking modes in Docker?**

* Docker provides several networking modes that allow you to configure how containers communicate with each other and the host machine.

1. **Bridge networking mode**:

* This is default networking mode for docker container.
* Containers run on a private network, and docker creates a bridge interface to connect them.
* Containers can communicate with each other using IP addresses.
* Host machine can access containers using IP addresses.

1. **Host networking mode**:

* containers share the host machines network stack.
* Containers can access the host machines network interfaces.
* No isolation between containers and host machine.

1. **None networking mode**: containers run without any network interface. Containers have no network connectivity. Useful for containers that don't require network access.
2. **Overlay networking mode:**

* allows multiple docker daemons to communicate with each other.
* Enables docker swarm mode and multi host networking.

1. **Macvlan networking mode:**

* allows containers to have their own MAC addresses and IP addresses.
* enables more advanced networking configurations.

1. **IPvlan networking mode**:

* like macvlan, but uses IP addresses instead of MAC addresses.

1. **Container networking mode**:

* allows a container to join another containers network.
* Containers can communicate with each other using IP addresses.

1. **Custom networking mode**:

* Allows users to create custom networks using third party plugins.

19. **How do you expose ports in a Docker container?**

* Exposing ports in a docker container allows applications running inside the container to communicate with the outside world or with other containers.
* Using the –p flag with docker run. (docker run –p host\_port: container\_port image\_name). Docker run –p 8080:80 nginx.
* Docker run –p image\_name
* Expose port\_number

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

|  |  |  |
| --- | --- | --- |
| Feature | Docker stop | Docker kill |
| Purpose | Docker stop command allows the container to terminate gracefully. | Docker kill command terminate it immediately |
| Signal sent | SIGTERM (graceful shutdown) | SIGKILL (immediate termination) |
| Timeout | Waits for a default timeout (10 seconds) | No time. Stops immediately. |
| Use case | Allows cleanup and graceful shutdown. | Forces immediate stop without cleanup. |
| behavior | Container can perform shutdown tasks. | Container is terminated instantly. |
| command | Docker stop <container name or container id> | Docker kill <container id or container name> |