**Assignment**

* What is Docker, and why is it used?
* Docker is an Open-Source platform designed to simplify and streamline the process of building, deploying and managing the applications using the containerization.
* These containers include the application and all the dependencies, ensuring it runs consistently accross different environments.
* Docker provides a lightweight alternative to Virtual Machines(VM's) by using the host system's OS kernel while isolating the applications in seperate containers.
* This improves efficiency, scalability and portability.
* **Uses:**
* **Consistency accross Environments:**
* Containers include everything that an Application it needs, ensurig it the same way regardless of where it's deployed, be it is your local machine, production environment and testing environment.
* **Efficient Resource Usage:**
* Docker containers share the host OS kernel making them light weight, fast to start and less resouce intensive.
* **Simplified Development and Deployment:**
* Dockers makes it easy to create reproducible developement environments. Teams can collaborate more effectively because they work in identical setups.
* **Poratability:**
* Containers can run on any system taht supports Dockers, whether it a developer's laptop, an on-premise data center, or cloud servers like AWS,Azure or GCP.
* **Scalability:**
* Docker works seamlessly with containerisation tools like Kubernetes, enabling easy scaling and management of containerized apllications.
* **Dependency Management:**
* Eliminates it "works on my machine" by packaging dependencies inside the container.
* **Supports Microservices:**
* Ideal for breaking the application into smaller and independent services.
* **Improved CI/CD:**
* Automates s/w development and testing with seamless integration.
* How is Docker different from a virtual machine (VM)?
* **Docker:**
* Uses host OS kernel, running applications in isolated environments.
* Lightweight and fast since it shares the host OS kernel.
* Containers are small and starts in seconds.
* Efficient, consumes fewer resources as containers shares same OS kernel.
* Highly portable, runs the same way accross different platforms.
* Process level isolation.
* Easy to scale up/down quickly by using container orchestration.
* Ideal for microservices, cloud deployments, CI/CD, and lightweight apps.
* **Virtual machine (VM):**
* Runs a full guest Os on top of hypervisor.
* Slower due to full OS virtualization for each VM.
* VM are larege and takes minutes to boot.
* High resource usage because each VM as it's own Os and kernel.
* Less portable, requires configuration for different OS and environments.
* Stronger isolation, as eachVM runs a seperate OS.
* Scaling is slower due to its OS overhead.
* Best for running different OS environments, legacy applications, or when strong isolation is needed.
* What are the main components of Docker?
* The main components of Docker:
* **Docker Engine:**
* The core of Docker, it consists of 3-main parts:
* **Docker Daemon:** runs on the host machine and manages Docker objects.
* **Rest API:** allows communication between Docker Daemon and other Docker components or external tools.
* **Docker CLI:** a command - line interface for users to interact with Docker.
* **Docker Images:**
* Lightweight, immutable snapshots that contain everything needed to run an application.
* Docker containers are instantiated from these containers.
* **Docker Containers:**
* These are runnable instances of Docker Images.
* They are isolated environments where applications and their dependencies run.
* **Docker Hub:**
* A Cloud - based registry for Docker images.
* It allows users to share,pull and manage images.
* You can find pre built images images or push your own-custom builts.
* **Dcoker Compose:**
* A tool to define and run multi conatiner applications using a YAML file.
* Great for managing complex setups, such as microservices in a simplified ways.
* **Docker Swarm:**
* Docker's native orchesthration tool for managing a cluster of Docker hosts, enabling the deployment and scaling of containers accross multiple machines.
* **Docker Volumes:**
* Persistent storage solutions for sharing data between containers or keeping intact when conttainers are stopped.
* **Docker Networking:**
* Provides isolated networking environments for containers to communicate with each other or the external world.
* Key options include bridge, host, overlay, and custom networks.
* Explain the difference between Docker images and Docker containers.
* **Docker Images:**
* Docker images are read-only templates that contain the blue print for creating containers.
* They include the application, dependencies, libraries, configurtions and its OS layers.
* Images are immutable, menaing they cannot be changes once created.
* It defines how should be the container built and what it should contain.
* Docker images ar built by using Docker file, which is a text file with instructions on how to assemble the image.
* Docker images are stored in registeries like Docker Hub or Private repositories.
* **Docker Containers:**
* Docker containers are running instances of Docker images.
* They are the actual execution environments where applications run.
* Dockers are mutable can be started, stopped and can be modified at runtime.
* Containers provide isolated environments for applications, sharing the host OS kernel.
* Containers are ephemeral.
* They can be started, stopped and cen be restarted as needed.
* What is a Dockerfile?
* **Docker File:**
* Docker file is a text file that contains a set of instructions to build a Dockr Image.
* It acts as a blue print for creating a containerized environment, specifying everything that the application needs to run.
* **Key Features:**
* **Base Image:** specifies the starting point of your image.
* **Commands and Instructions:**
* Defines the speps to install the dependencies, copy files, set environment variables, expose ports and run the application.
* **FROM:** specifies the base image
* **WORKDIR:** sets the working directory inside the conatiner.
* **COPY:** copies files from the host machine into the container.
* **RUN:** executes the commands inside the container during the build.
* **CMD:** specifies the default command that runs when the container starts.
* **EXPOSE:** defines the port that container will listen on
* **ENTRYPOINT:** similar to CMD, but makes the container behave like an executable.
* **ENV:** sets the envoironment variables inside the container.
* What command is used to build a Docker image?
* **docker pull image name:** to pull the docker image from the official website
* **docker images:** used to check for the images created.
* How do you run a container from an image?
* **docker run -itd ---name "Yourname" -p "port number" "your image id"**
* How do you list all running containers?
* **docker ps -a**
* What command is used to stop a running container?
* **docker stop "container id or container name"**
* How do you remove a Docker container?
* First we need to stop the container and then remove the container:
* **docker stop "container id or container name"**
* **docker em "container id or container name"**
* **Force stop:** if don't wanna stop and remove the container directly:
* **docker rm -f "container id or container name"**
* What is the difference between CMD and ENTRYPOINT in a Dockerfile?
* **CMD:**
* Specifies the default command or arguements that will be execited when the container starts.
* If a command is provided when running the container, it overrides the CMD instruction.
* Best for providing the default values or fallback commands.
* **ENTRYPOINT:**
* Defines the command that will always run, regardless of ant additional commandsprovided during docker run.
* Arguements are provided at runtime are appended to the ENTRYPOINT command rather than replacing it
* Best for ensuring a primary executable
* What is a Docker volume, and why is it used?
* A Docker volume is a mechanism for storing and persisting the data generated by the containers.
* Unlike the ephemeral nature of containers, volumes enable data to persist independently of the container lifecycle.
* **USES:**
* **Data Persistence:**
* Volumes ensure that there is no loss of data when the container has been stopped, removed or restarted.
* **Container Independence:**
* Volumes exist outside the container itself and are managed by docker.
* This allows multiple containers to share the same data.
* **Improved Performance:**
* Volumes are optimized for dockers and typically perform better than the storing the data directly inside the container's writable layer.
* **Ease of Backup and Restore:**
* Sice volumes are managed by docker, they are easy to backup, restore or move between the hosts.
* **Seperation of Concerns:**
* By storing data in volumes, you can keep the application code inside containers separate from the data, promoting modularity and easier maintenance.
* How do you persist data in Docker containers?
* Containers are ephemeral by nature - data inside the containers will be lost once the container was stopped, removed and restarted.
* To ensure data persists, Docker provides 2 solutions:
* **Docker Volumes:**
* Managed by Docker and stored on the host filesystem, outside of the container's lifecycle.
* Allows data to persist independently of the container.
* They are easy tro share between multiple containers.
* ideal for data base storage, logs etc.
* **Bind Mounts:**
* Maps a specific directory from the host to the container.
* Bind mounts give containers direct access to the host's directory.
* You can specify the exact path on the host you want to bind.
* Best when you need the container to work with existing data on the host.
* What is a Docker Compose file? How is it used?
* It is a YAML file.
* It simplifies the definition and management of multi-container docker applications.
* Allows you to configiure the application services in a single file, makes it easier to define relationships, networks and volumes for these services.
* **Way to use:**
* Create a file and save it as **docker -compose.yml** in the root directory of the project.
* **Start the Services:**
* Run the command to build and start the defined services:
* **"docker -compose up"**
* **Stop the Services:**
* Stops all the running services:
* **"docker -compose down"**
* **Background Execution:**
* Runs the services in the detacthed mode:
* **"docker -compose up -d"**
* How do you scale services using Docker Compose?
* Scaling services in Docker Compose is a way of running multiple instances of a service to handle the overload issues or better fault tolerence.
* **Steps to Scale Services:**
* **Define the Service:**
* Ensure that the service is define properly, including the dependencies, environment variables and dependencies.
* **Run the Scaling Command:**
* Use the --scale flag to specify the number of instances (replicas) for a particular service
* **docker -compose up --scale "service name" = "no.of replicas"**
* **Background Execution:**
* Runs the services in the detacthment mode for better management:
* **docker -compose up --d "service name" = "no.of replicas"**
* **Verify Scaling:**
* Check for running containers:
* **docker ps**
* How do you check the logs of a running container?
* **Identify the Container name or Id:**
* **docker ps**
* **View logs:**
* **docker logs "Container name or Id"**
* **Follow logs in real time:**
* **docker logs -f "Container name or Id"**
* **Limit logs Output:**
* **docker logs --tail "number" "Container name or Id"**
* **View Timestamps:**
* **docker logs --timestamps "number" "Container name or Id"**
* What is the purpose of the .dockerignore file?
* Used to specify files and directories that should be excluded while creating a Docker Image.
* Purpose is to improve the build process by ensuring that unnecessary files are not added to the Docker Image, which can:
* **Reduce Image Size**
* **Speed up Build Times**
* **Protect Sensitive Info**
* **Ensure Clean Environments**
* What are the different networking modes in Docker?
* **Bridge:**
* Each container connected to a private virtual network created by Docker.
* Can communicate with each other through this bridge network.
* You can also map the container to the host to enable external access.
* **Host:**
* The container shares the host's network namespace, meaning it has direct access to the host's network interfaces.
* This eliminates network isolation, allowing better performance for certain use cases.
* **None:**
* The container has no network access.
* This mode is useful for more secure purposes, when no need of network connectivity.
* **Container:**
* Allows one container to share the network namespace of another container.
* The second container uses the same IP address and network settings as the first container.
* **Overlay:**
* Used in a Docker Swarm or Kubernetes environment.
* Allows multiple Docker daemons to communicate securely over a virtual network.
* This is ideal for connecting services in a distributed system.
* How do you expose ports in a Docker container?
* **Using Dockerfile:**
* Using EXPOSE keyword
* **EXPOSE 8080**
* **At Runtime:**
* **docker run -p 8080:8080 image name**
* **Publishing all the Exposed ports:**
* **docker run -P "image name"**
* **Inspaecting the published ports:**
* **docker ps**
* What is the difference between docker stop and docker kill?
* **Docker stop:**
* Sends the SIGTERM signal to the main process running inside the container, giving it time to clean up resources
* It gives some time to clean up.
* Suitable for scenarios where the containerized application needs to perform shutdown procedures before termination.
* **docker stop "Container name or Id"**
* **Docker Kill**:
* Immediately sends the SIGKILL signal to the container’s main process, stopping it without allowing any cleanup.
* The container is stopped instantly without waiting for any process to finish.
* Useful when a container is unresponsive or you need it terminated immediately.
* **docker kill "Container name or Id"**