Tutorial 3: Docker and Dockerfile

Question Set:

Q1. What is Docker? Discuss the motivations of using Docker technology.

ANSWER:
Docker is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating system kernel and therefore use fewer resources than virtual machines.

Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

Q2. What are Linux namespaces and cgroups, and how do they work together to enable containerization?

ANSWER:

- **Namespaces** are a feature of the Linux kernel that provides process isolation. They achieve this by giving a group of processes its own virtualized view of system resources, such as the network, filesystem, and process IDs, effectively creating a private operating environment.
- **Cgroups (Control Groups)** are a Linux kernel feature that manages and limits the resource usage of processes. They are used to enforce constraints on how much CPU, memory, and disk I/O a specific group of processes can consume from the host system.

- Namespaces build the container's walls, isolating it from the host and other containers so it cannot see or interact with outside processes.
- Cgroups then determine the resources available within those walls, limiting how much CPU and memory the containerized process can use to ensure it doesn't disrupt the host. In essence, namespaces provide isolation, while cgroups provide resource limitation.

Q3. What are Docker Image, Docker Container, and Docker Registries? Please give brief descriptions of these concepts.

ANSWER:

- **Docker Image:** An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For

example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run. You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

- **Docker Container:** A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state. By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine. A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.
- **Docker Registry:** A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry. When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.

Q4. What is Dockerfile? Please discuss what the good practices for writing Dockerfiles are.

ANSWER:
Dockerfile is a text file that contains a collection of instructions and commands for building a docker image and running as a container.

Some good practices for writing Dockerfiles:

- Use ".dockerignore" to exclude files not relevant to the build (without restructuring your source repository) use a .dockerignore file. This file supports exclusion patterns similar to .gitignore files.
- Use multi-stage builds to allow you to drastically reduce the size of your final image, without struggling to reduce the number of intermediate layers and files. Because an image is built during the final stage of the build process, you can minimize image layers by leveraging build cache.
- Don't install unnecessary packages. To reduce complexity, dependencies, file sizes, and build times, avoid installing extra or unnecessary packages just because they might be "nice to have." For example, you don't need to include a text editor in a database image.
- Decouple applications. Each container should have only one concern. Decoupling applications into multiple containers makes it easier to **scale horizontally** and **reuse containers**. For instance, a web application stack might consist of three separate containers, each with its own unique image, to manage the web application, database, and an in-memory cache in a decoupled manner. Limiting each container to one process is a good rule of thumb, but it is not a hard and fast rule. Use your best judgment to keep containers as clean and modular as possible. If containers depend on each other, you can use Docker container networks to ensure that these containers can communicate.

- Minimize the number of layers. Only the instructions RUN, COPY, ADD create layers. Other instructions create temporary intermediate images, and do not increase the size of the build. Try to use "&&" to include all the possible RUN instructions in the same line. Where possible, use multi-stage builds, and only copy the artifacts you need into the final image. This allows you to include tools and debug information in your intermediate build stages without increasing the size of the final image.
- and more…. Please read [2] for more good practices for writing Dockerfiles.

Q5. How to make the image built from Dockerfile as lean as possible?

ANSWER:

- Import a base image with a smaller footprint
- Install only necessary dependencies
- Use clean-up commands.
- Multi-stage builds

References

[1] Docker documentation. https://docs.docker.com/get-started/overview/.
[2] Best practices for writing Dockerfiles. https://docs.docker.com/develop/develop-images/dockerfile_best-practices/