**APPLICATION CONTAINERIZATION AN D ORCHESTRATION GROUPS**

**LAB EXPERIMENT-1**

**Name: Aanchal Tailwal**

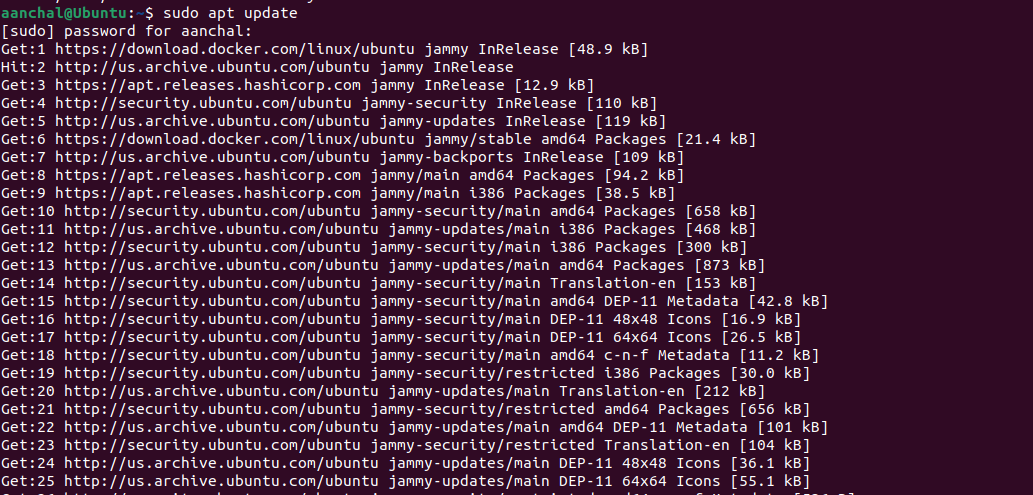
**Branch: BTech CSE**

**Batch: DevOps B-4**

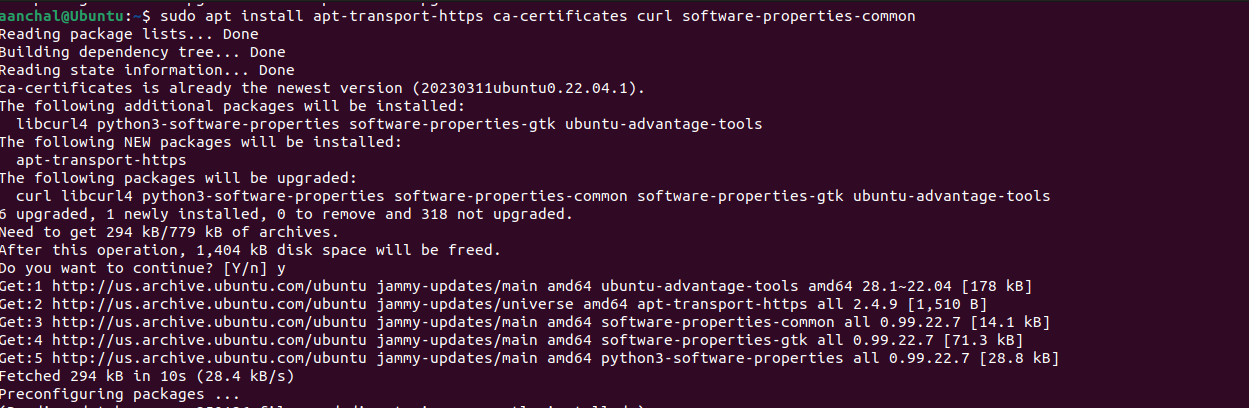
**SAP ID: 500097386**

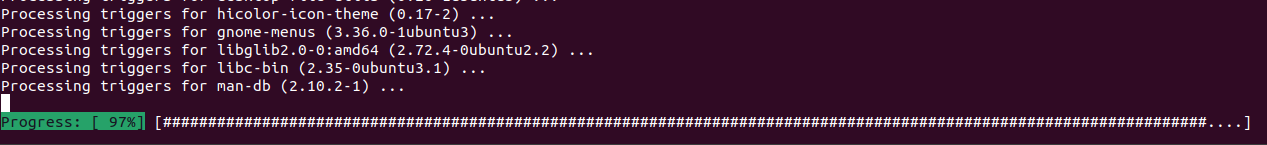
**QUES 1. Discuss installation steps of Docker with proper screenshots.**

1. **Update Package Repositories:**

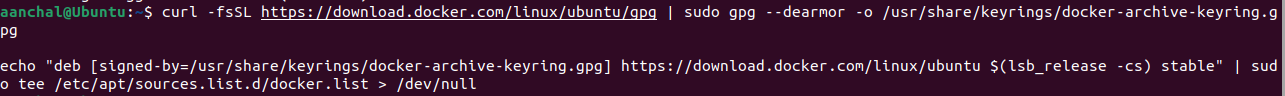


1. **Install Prerequisites:**

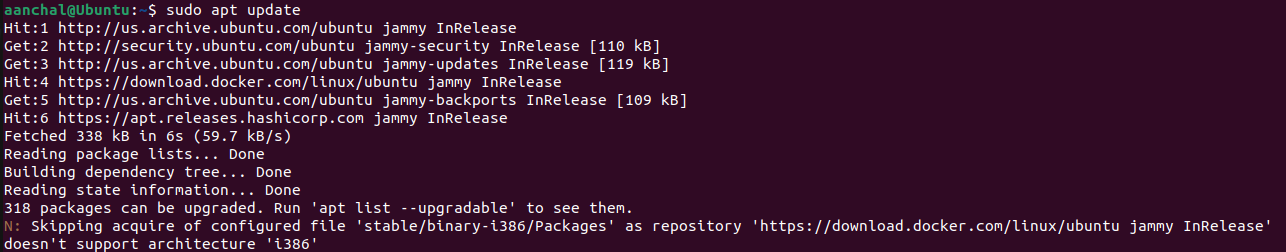




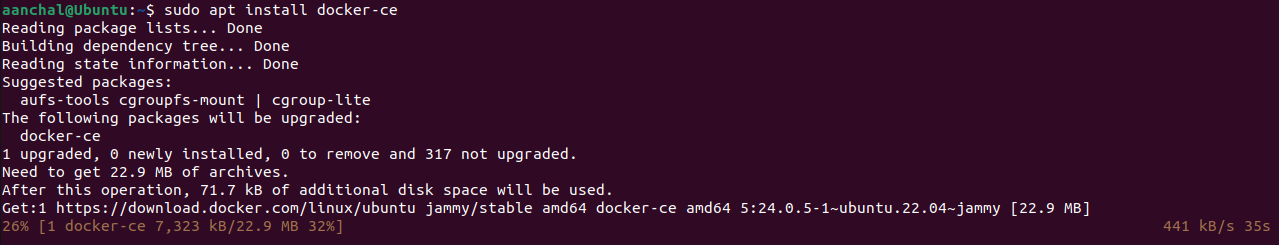
1. **Add Docker Repository:**



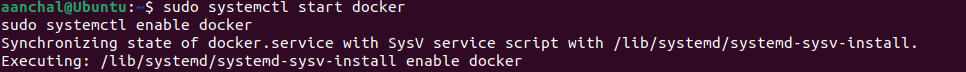
1. **Update Repositories**



1. **Install Docker Engine**



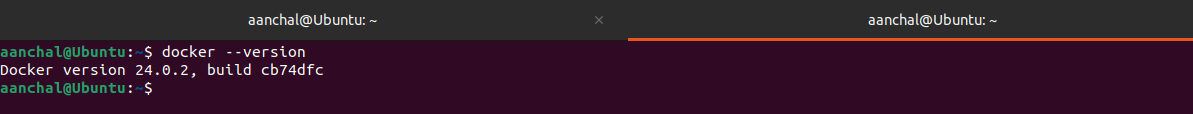
1. **Start and Enable Docker**



1. **Add user to Docker Group**



1. **Verify Installation**



**QUES 2. Define Docker architecture/platform for containerizing any application.**

Docker is a platform and a set of tools designed to simplify the process of creating, deploying, and running applications using containers. Containers are lightweight, portable, and self-sufficient units that can package an application and its dependencies, enabling consistent deployment across different environments.

Here's an overview of the key components and concepts in Docker's architecture:

1. Docker Engine: The Docker Engine is the core component responsible for running and managing containers. It includes:
2. Docker Daemon: The background service that manages the containers on a host system. It handles container lifecycle, image management, and network communication.
3. Docker Client: A command-line tool or API that interacts with the Docker Daemon to build, manage, and control containers.
4. Images: Docker images are read-only templates that contain everything needed to run an application, including the code, runtime, system tools, libraries, and settings. Images are created from a set of instructions in a Dockerfile, which defines the application's environment.
5. Containers: Containers are instances created from Docker images. They encapsulate an application along with its runtime environment, libraries, and settings. Containers are isolated from each other and from the host system, making them highly portable and consistent across different environments.
6. Docker Registry: A Docker registry is a repository for storing and distributing Docker images. The most well-known Docker registry is Docker Hub, but organizations often set up their private registries for security and control. Images can be pulled from and pushed to registries.
7. Docker Compose: Docker Compose is a tool for defining and running multi-container applications using a simple YAML file. It allows you to define the services, networks, and volumes required for your application and manage them as a single unit.
8. Docker Swarm (Optional): Docker Swarm is Docker's native orchestration solution for managing a cluster of Docker nodes (machines). It enables you to create and manage services across multiple nodes, ensuring high availability and scalability.
9. Docker CLI: The Docker Command-Line Interface (CLI) is used to interact with the Docker Engine. It provides a set of commands for building, running, managing, and inspecting containers and images.

**QUES 3. Discuss steps for containerizing python application**

Containerizing a Python application involves packaging the application code, dependencies, and runtime environment into a Docker container. Here are the steps to containerize a Python application:

1. Install Docker: Make sure Docker is installed on your system. If not, follow the installation steps outlined in the previous response.
2. Create a Dockerfile: A Dockerfile is a script that contains instructions for building a Docker image. Create a file named Dockerfile in your application's root directory. Here's a simple example for a
3. Build the Docker Image: Open a terminal, navigate to your application's directory containing the Dockerfile, and run the following command to build the Docker image:

* docker build -t my-python-app .

Replace my-python-app with a suitable name for your image.

1. Run the Docker Container: Once the image is built, you can run a container based on that image. Use the following command:

* docker run -it my-python-app

This will start a container from the my-python-app image and run the command specified in the Dockerfile's CMD instruction.

1. Persist Data (Optional): If your application generates or uses data that should be persisted beyond the container's lifespan, you can use Docker volumes to mount host directories into the container.

* docker run -it -v /host/path:/container/path my-python-app

Replace /host/path and /container/path with the appropriate paths.

1. Publish to Docker Hub (Optional): If you want to share your Docker image with others or deploy it to different environments, you can push it to a Docker registry like Docker Hub. To do this, you need to create an account on Docker Hub, tag your image with your Docker Hub username, and push it to the repository.

* docker tag my-python-app your-docker-hub-username/my-python-app
* docker push your-docker-hub-username/my-python-app

Regenerate