Introduction

From a user or human perspective, the primary purpose of an operating system (OS) is to run programs. Installing an operating system requires physical hardware components such as RAM, CPU, network card, and a hard disk. There are four different methods for installing an operating system: Bare metal, Virtualization, Over the cloud, and Containerization.

Containerization and Its Significance

Containerization, exemplified by tools like Docker, stands out for its ability to launch an entire OS in less than a second. In today's agile world, businesses thrive on speed, and this rapid OS provisioning becomes crucial for tasks such as building, testing, and deployment.

Why OS Provisioning in 1 Second?

Agile Development and Time to Market:

- Businesses depend on the speed of time to market. Rapid OS provisioning allows quick development, testing, and deployment of programs, meeting the demands of the agile world.

Complete Environment Launch:

- Containerization facilitates the launch of the complete OS environment within seconds, a process known as OS provisioning.

Role of Docker in OS Provisioning

To achieve OS provisioning in seconds, Docker, a containerization tool, is utilized. In the containerization world, the OS or environment is referred to as a container. Docker works on the concept of containerization and operates on Linux as the base OS.

Installing Docker on AWS Cloud

Launch Docker Host on AWS:

- Provide a container name and choose the Amazon Linux image.
- Optionally, create a key pair or proceed without a key.
- Connect to AWS Linux operating system.

Install Docker:

- Switch to the root user with `sudo su -root`.
- Install Docker using `yum install docker`.
- Start Docker service with `systemctl start docker`.

Managing Docker:

- Use `docker ps` to check running containers.
- Use `docker images` to view available Docker images.

Working with Docker Containers

Pulling and Running Images:

- Pull Ubuntu image with `docker pull ubuntu:14.04`.
- Launch a new OS with `docker run -i -t ubuntu:14.04`.

Container Interaction:

- Interact with the newly launched container.
- Use `docker ps -a` to check all containers in running and shut-down states.

Container Management:

- Stop a container with `exit`.
- Start a shut-down container with `docker start <container_id>`.
- Attach to a running container with `docker attach <container_id>`.
- Use CTRL+p+q to go back to the base OS without shutting down the container.

Advanced Container Launch:

- Use `-d` to launch a container in detach mode.
- Assign a name to a container with `--name` option.

Network Setup and Connectivity:

- Docker automates network setup for containers.
- Check network connectivity with `ping <ip_address>`.

Managing Docker Containers and Custom Images

Stopping and Removing Containers

Stop Running Container:

- Use `docker stop <container_name_or_id>` to halt a running container.

Remove Container:

- Use `docker rm <container_name_or_id>` to delete or remove a container.

Software Installation in Operating Systems

Limited Software Commands:

- Operating systems come with limited software or commands pre-installed.
- For example, the `ifconfig` command might not be available in some Linux distributions.

Installing Software:

- Identify the software providing the required command using `yum whatprovides <command_name>`.
- Install the required software, e.g., `net-tools` for the `ifconfig` command.

Bulk Container Operations:

- Stop all containers at once: `docker stop $(docker ps -q)`.
- Remove all containers at once: `docker rm -f $(docker ps -q)`.

Custom Images and Containerization

Launching Containers from Images:

- Containers can only be launched from images, and all software within a container comes from the image.

Creating Custom Images:

- Develop a custom image with specific software requirements for a product.

Using the Commit Technique

Launch Container and Install Requirements:

- Launch a container (e.g., from the CentOS image) with a specified name (e.g., os1).
- Install necessary requirements within the container.

Convert to Custom Image:

- Execute `docker commit <container_name> <custom_image_name>` to create a custom image (e.g.,ayush).

Launching Container with Custom Image:

- Launch a container using the custom image, observing that pre-installed requirements are present.

Using Dockerfile

Overview of Dockerfile:

- A Dockerfile is a text document containing commands to assemble an image.
- Offers a more flexible and reproducible approach compared to the commit technique.

Creating Custom Image with Dockerfile:

- Write scripts using Dockerfile keywords to specify the desired configuration.
- Allows making small changes without recreating the entire image from scratch.

Advantages of Dockerfile:

- Easy modification: Small changes can be incorporated by updating the Dockerfile.
- Improved reproducibility: Any small adjustment can be made by editing the file.

Creating a Dockerfile for Custom Image

Step 1: Folder and Dockerfile Creation

Create a Folder:

- Open a terminal and create a folder for your Docker project using `mkdir <folder_name>`.

Create Dockerfile:

- Inside the newly created folder, use the `vim` command to create a file named `Dockerfile`. Use `vim Dockerfile`.

Step 2: Dockerfile Keywords

FROM Keyword:

- Use the `FROM` keyword to specify the parent image from which you are building.
- Example: `FROM centos:7` (This uses the CentOS image as the base).

RUN Keyword:

- Utilize the `RUN` keyword to execute commands in a new layer on top of the current image.
- Example: `RUN yum install -y net-tools` (This installs the net-tools package).

Step 3: Building the Image

Build the Image:

- Run the following command to build the custom image from the Dockerfile:

  docker build -t <custom_image_name> <path_to_Dockerfile>

  Example: `docker build -t myimg .` (Assuming Dockerfile is in the current directory).

Step 4: Checking the Custom Image

Check Custom Image:

- After successful build, use the following command to view the list of Docker images.

  docker images

Verify Custom Image:

- Look for your custom image in the list, identified by the specified `<custom_image_name>`

Exemplar Dockerfile

Below is an exemplar Dockerfile for creating a custom image based on CentOS with the installation of the `net-tools` package:

```
# Use the latest CentOS image as the base
FROM centos:7

# Install the net-tools package
RUN yum install net-tools -y
```

Save this Dockerfile in the folder created in Step 1.
docker build –ulimit nofile=1024000:1024000 -t myweb:v .

*The `--ulimit nofile=soft_limit:hard_limit` option helps control the file descriptor limits during the Docker image build process, ensuring that the build doesn't consume too many resources.