Khaula Molapo

Operating System Week 12

Hands-On Virtualization Lab

For this lab, a Type 2 hypervisor called VirtualBox was installed on a personal computer. VirtualBox allows users to create and manage virtual machines that run different operating systems inside the main OS.

1. Creating a Virtual Machine

After installing VirtualBox, a new virtual machine was created and named 'Ubuntu Server'. The operating system type was set to Linux, and the version was set to Ubuntu (64-bit).

2. Virtual Machine Configuration

The following settings were used:

- • CPU: 2 virtual cores
- • Memory: 2048 MB (2 GB) RAM
- • Virtual Hard Disk: 20 GB (dynamically allocated)
- • Network: NAT mode for internet access

3. Installing Ubuntu Server

An Ubuntu Server ISO file was downloaded and mounted as the startup disk. The VM booted from the ISO, and the Ubuntu Server installation process was followed step-by-step. After installation, the VM successfully booted into Ubuntu's command-line interface.

4. Taking and Reverting a Snapshot

After setup, a snapshot named 'Fresh Install' was created in VirtualBox. Then, a change was made inside the VM (for example, installing Apache using 'sudo apt install apache2'). After confirming the change, the snapshot feature was used to revert the VM back to its clean state. This showed how snapshots help restore a virtual machine to a previous working condition.

Snapshots are useful for testing, because they let you experiment with system changes without any permanent risk.

Containers vs. Virtual Machines Analysis

Both containers and virtual machines (VMs) are technologies used to run multiple applications or environments on one physical system. However, they work in different ways and are used for different purposes.

1. Isolation Level

Virtual machines provide full isolation because each VM runs its own complete operating system with its own kernel. Containers, on the other hand, share the host OS kernel but keep applications isolated at the process level. This makes containers lighter but slightly less isolated than VMs.

2. Startup Time and Overhead

Containers start much faster because they don't need to boot a full OS—just the application environment. VMs take longer to start since they must boot an entire operating system. Containers use fewer system resources, while VMs consume more CPU, RAM, and storage due to running full guest operating systems.

3. Portability

Containers are highly portable because they include all dependencies needed to run an app. You can easily move a Docker container between different systems as long as Docker is installed. VMs are less portable since moving them requires copying larger image files and ensuring the hypervisor supports them.

4. Use Case Comparison

You would choose virtual machines when you need full OS-level isolation, such as running different operating systems or testing system-level software. VMs are also ideal for scenarios that require strong security separation between environments.

Containers are a better choice when you need fast startup, scalability, and easy deployment—like hosting web applications, microservices, or development environments. Containers allow developers to build once and run anywhere with minimal configuration.

Conclusion

In summary, containers and VMs both help improve resource usage and flexibility, but they serve different purposes. Containers are lightweight, fast, and portable, while VMs provide stronger isolation and flexibility for running multiple OSes. Choosing between them depends on whether performance or isolation is more important for your use case.