Bahir Dar University

Department of Software Engineering

Operating System and System Programming

Individual Assignment

Section B

Name: Bezawit Amogne
Id:1601081
Course: Operating System and System Programming

Submission date: 6/09/2017

Submitted to Lec. Wondmu

Introduction

An Operating System (OS) is a software that sits in between a computer hardware and the user. It is a user-friendly interface that controls the computer's hardware resources like the CPU, memory, storage devices and input/output peripherals.

The name for this operating system may still be rarely heard in the ears of the master linux world, even in the ears of beginners though, but who would have thought the user is now increasingly in interest in developed countries to developing countries such as Indonesia. Endless OS is a new OS for PC / notebook platform, developed by Endless Computer using Linux base.

Endless Mobile, Inc. developed the Endless OS-based operating system and hardware reference platform for it. The company was founded in 2011 and is based in San Francisco, California with additional offices in Rio de Janeiro, Brazil.

Endless Mobile, Inc.

Mengetik	Pribadi
Industri	Perangkat keras komputer Perangkat lunak komputer Elektronik konsumen Distribusi digital Manufaktur tanpa pamrih
Didirikan	2011 ; 7 tahun yang lalu , di San Francisco , California , AS
Markas besar	San Francisco , California , Amerika Serikat
Daerah yang dilayani	Di seluruh dunia
Orang kunci	Matt Dalio (CEO) [1] Marcelo Sampaio (CGO) Richard Vignais (CDO) Jonathan Blandford (VP, Teknik)
Produk	OS tanpa akhir [2] · Mini tanpa akhir [3] · Tak ada habisnya [4] · Misi Tanpa Akhir [5]
Situs web	endlessos .com

History - Endless Os

Endless was founded in May 2012 which is housed in San Francisco, California by Matthew Dalio and Marcelo Sampaio. In the first three years, the company focused solely on designing through fieldwork in Rocinha, favela, to the largest in Rio de Janeiro, Brazil, as well as in Guatemala.

In April 2015, the product was launched to the general public through a campaign on the crowdfunding platform Kickstarter. it did not take long to collect $ 176,538 with 1,041 supporters, enough in less than 30 days, incredible statistics.

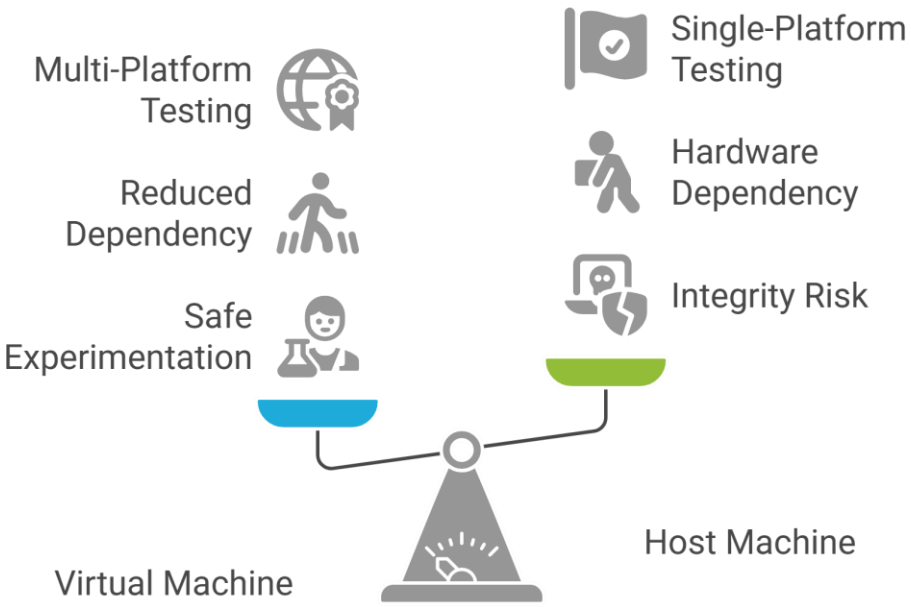
what is Endless Os?

Endless OS is a Debian Linux derivative, built on Linux kernel and other open source technologies (Chromium, GNOME, GRUB, GTK, PulseAudio, Rufus, Systemd, XOrg, Yelp, and more). Unlike most Linux distributions, it uses the read-only root system files managed by OSTree and Flatpak for sending and updating all of its applications, Its user interface is also based on a highly modified GNOME desktop environment

Background

In the modern era of computing, virtualization technology has revolutionized the way software developers, IT professionals, and hobbyists interact with operating systems. Virtual environments allow users to run multiple operating systems simultaneously on a single hardware machine, enabling flexible, cost-effective, and efficient workflows. Tools such as VMware Workstation and Oracle VM VirtualBox have emerged as indispensable platforms for testing, development, and education.

The installation of an operating system within a virtual machine offers numerous advantages. It provides a safe environment to experiment without risking the integrity of the host machine, reduces hardware dependency, and facilitates multi-platform compatibility testing. This capability is crucial for software engineering students and professionals, who must often test applications across diverse environments.



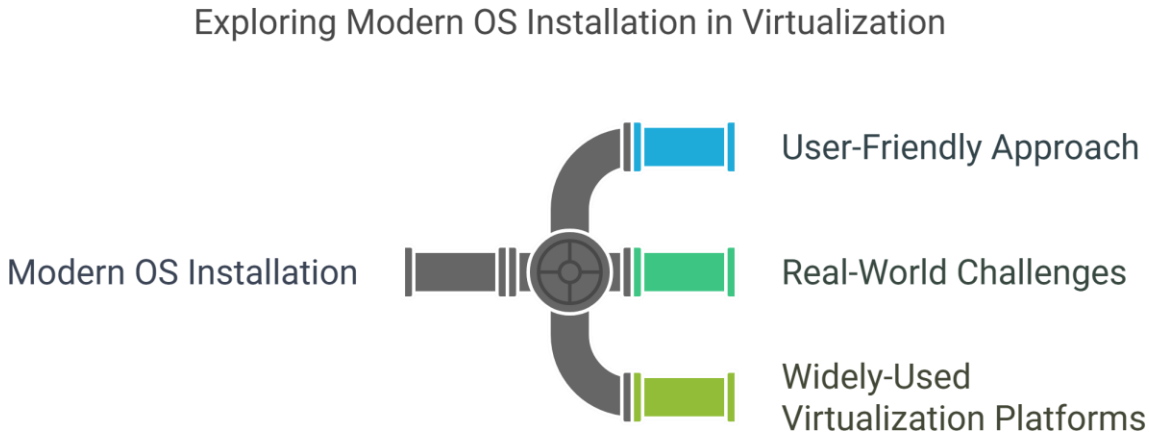
Virtual Machines Offer Safe and Flexible Testing

Motivation of Endless

The motivation for this project stems from two primary goals. First, as a software engineering student, mastering the use of virtualization tools is vital for academic success and professional growth. Virtual machines are a gateway to understanding system architecture, resource allocation, and compatibility challenges, providing invaluable hands-on experience in software development and IT operations.

Second, the need to streamline workflows and enhance productivity motivates this exploration. With the advent of Long-Term Support (LTS) operating systems like Ubuntu 22.04, developers can leverage a stable, reliable platform for sustained development efforts. Additionally, by documenting this process, the project seeks to empower peers and newcomers to confidently set up and utilize virtual environments, fostering a culture of collaboration and technical proficiency.

This paper focuses on the installation and setup of a modern operating system within widely-used virtualization platforms, emphasizing a user-friendly approach while addressing real-world challenges. It serves as both a learning opportunity and a practical guide for future endeavors in software engineering.



Objectives

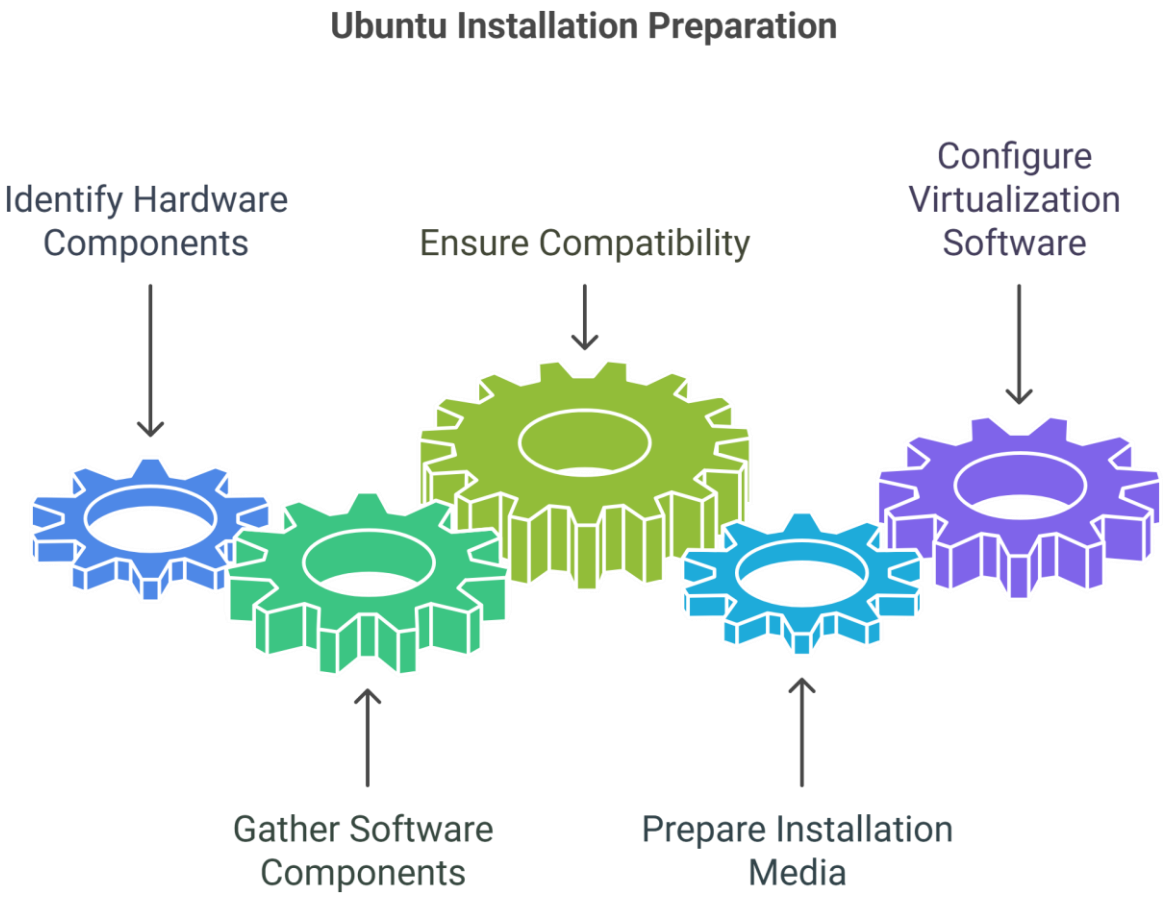
1. **Gain Hands-On Virtualization Skills**

a. Dive deep into VMware Workstation and Oracle VM VirtualBox to understand how hypervisors manage hardware resources, snapshots, and virtual networking.
b. Become comfortable creating, configuring, and launching VMs from scratch—with confidence to explore other virtualization platforms in the future.

2. **Install a Supported, Stable OS**
    a. Select and install a Long-Term Support (LTS) distribution—Ubuntu 22.04 LTS—to ensure security updates and compatibility for at least five years.
    b. Verify that the chosen OS runs smoothly on virtual hardware, with full driver support and access to guest-additions for seamless integration.
3. **Compare Virtualization Platforms**
    a. Document similarities and differences in setup, performance, and usability between VMware Workstation and VirtualBox.
    b. Measure basic metrics (e.g., boot time, RAM/CPU overhead) to highlight each tool's strengths and trade-offs for day-to-day development tasks.
4. **Develop Troubleshooting Expertise**
    a. Anticipate common installation hiccups—bootloader errors, guest-additions failures, network misconfigurations—and build a personal "debug toolkit" for quick resolution.
    b. Capture problem screenshots and log snippets, then methodically work through solutions so future you (and fellow students) can sidestep the same roadblocks.
5. **Understand Filesystem Options**
    a. Explore and evaluate filesystems available to Ubuntu guests (ext4, Btrfs, ZFS, etc.), then choose the best fit for a development VM.
    b. Document why ext4 (or your chosen filesystem) meets needs around performance, snapshot-friendliness, and data integrity in a virtual context.
6. **Create a Reusable, Peer-Ready Guide**
    a. Produce clear, step-by-step instructions enriched with annotated screenshots—so that any newcomer can replicate your setup in under an hour.
    b. Highlight "pro tips" and shortcuts to make the process smoother, fostering a collaborative spirit among classmates and open-source communities.
7. **Chart Future Improvements**
    a. Identify potential enhancements—such as automated scripting (Terraform, Vagrant), containerizing services (Docker on VM), or experimenting with other OS flavors—to extend this project beyond installation.
    b. Lay the groundwork for follow-on research into cloud-based virtualization, hybrid workflows, or advanced snapshot/version controls

Requirements

To ensure a smooth, reproducible installation of Ubuntu 22.04 LTS in both VMware Workstation and Oracle VM VirtualBox, you'll need to gather the following hardware and software components.



i. Hardware

- **CPU with Virtualization Extensions**
  - Intel® VT-x or AMD-V support enabled in your BIOS/UEFI

    - Multi-core processor (dual-core minimum; quad-core or higher recommended)
- **Memory (RAM)**
    - **Minimum:** 8 GB total on the host so you can comfortably allocate at least 4 GB to your VM
    - **Recommended:** 16 GB+ for running multiple VMs or more RAM-intensive workloads
- **Storage**
    - **Available Free Space:** ≥ 20 GB per VM
    - **Recommended:** Solid-state drive (SSD) for faster boot and disk I/O
- **Networking**
    - Stable internet connection for downloading ISOs, guest-additions, and system updates
    - Optional: Ethernet for reliable bandwidth, especially if working with large downloads
- **Graphics (Optional)**
    - Standard integrated GPU is fine for a headless or basic desktop VM
    - Discrete GPU only needed if you plan to run graphics-heavy applications inside the VM

ii. Software

- **Host Operating System**
    - Windows 10/11 (64-bit), recent Linux distributions (Ubuntu, Fedora, CentOS, etc.), or macOS Monterey and later
    - **Note:** Skip installing on deprecated host OSes without LTS or security support (e.g., Windows 7)
- **Virtualization Platforms**
    - **VMware Workstation**
        - Workstation Pro 16 or later, or Workstation Player 16+
    - **Oracle VM VirtualBox**
        - Version 6.1 or later
- **Guest OS ISO**
    - Ubuntu 22.04 LTS desktop or server image (downloadable from ubuntu.com)
- **Guest-Additions / Tools**
    - **VMware Tools:** for improved graphics, clipboard sharing, and drag-and-drop

    - **VirtualBox Guest Additions:** for shared folders, better display drivers, and seamless mouse integration
- **Utilities**
    - Archive manager (7-Zip, WinRAR, or native archive tools) to extract ISO files if needed
    - SSH client (PuTTY on Windows or built-in terminal on Linux/macOS) for headless VM access

Installation of endless operating system

Endless OS Installation: Step-by-Step Description
Requirements
A PC or laptop
At least 2GB RAM and 16GB storage
A USB flash drive (minimum 8GB)
Internet connection (for downloading the installer)

1. Download the Endless OS Installer

Visit the official website: https://endlessos.com/download/
Choose the version:
Basic (for internet-connected install)
Full (includes offline apps)
Download the .iso image or Windows Installer if using a Windows system.

2. Create a Bootable USB Drive

To install Endless OS on another computer:
Use a tool like balenaEtcher or Rufus.
Select the Endless OS ISO file.
Choose the USB drive.
Flash the OS onto the USB stick.
This USB will serve as your boot and installation media.

3. Boot from the USB Drive

Plug in the USB drive into your target PC.
Restart and enter the boot menu (usually F12, F10, ESC, or DEL).
Select the USB device as the boot source.
Endless OS will start loading its installation environment.

4. Choose Installation Option

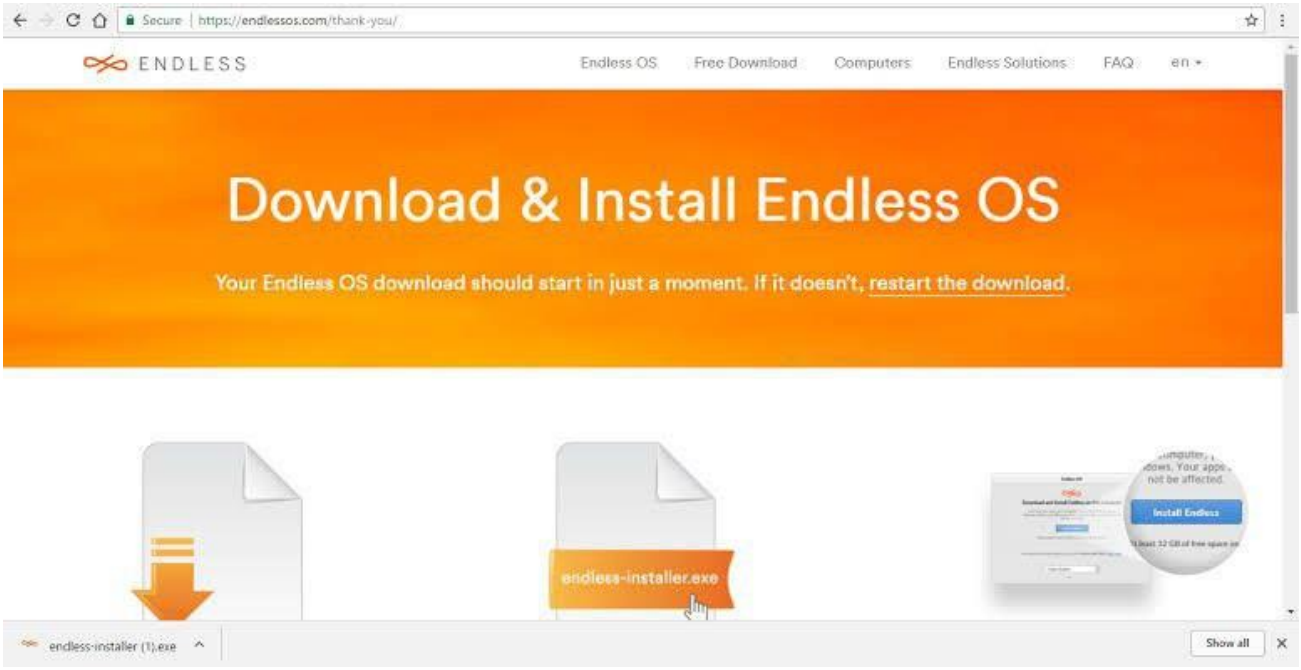
You'll be presented with two main choices:
Try Endless OS (runs it live from the USB without installing)
Reformat and Install Endless OS (erases current OS and installs Endless)
Choose "Reformat and Install" only if you're ready to replace the existing OS.

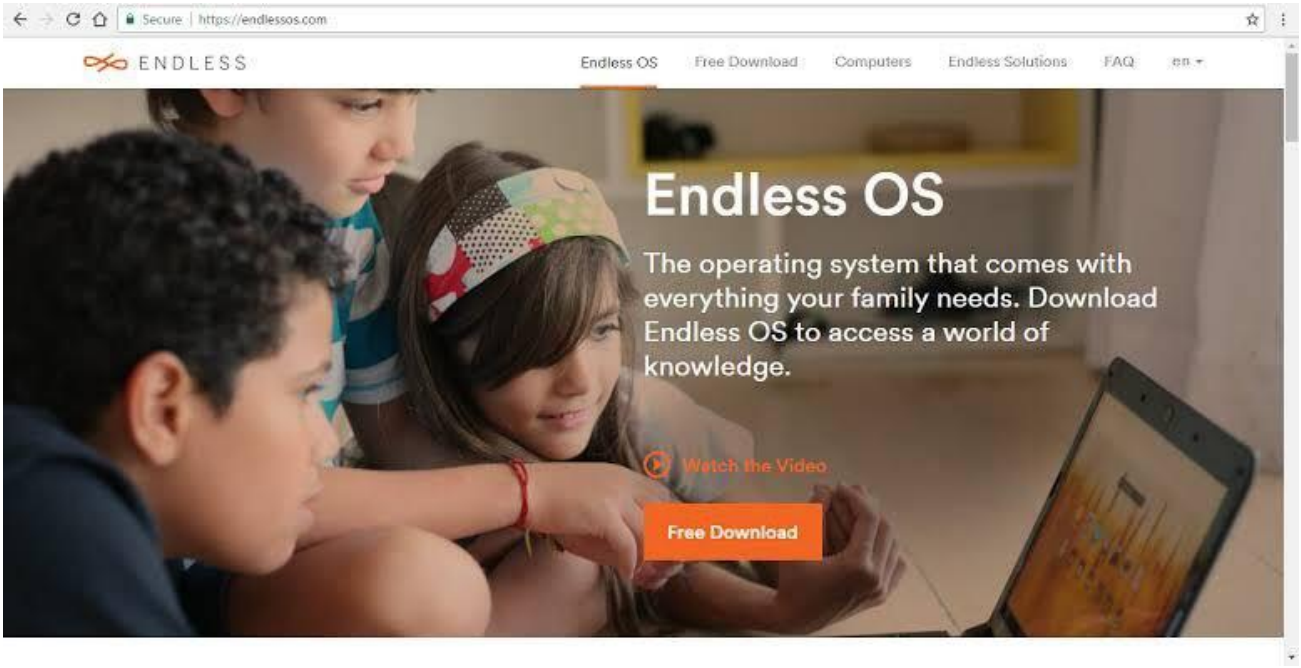
5. Configure Installation Settings

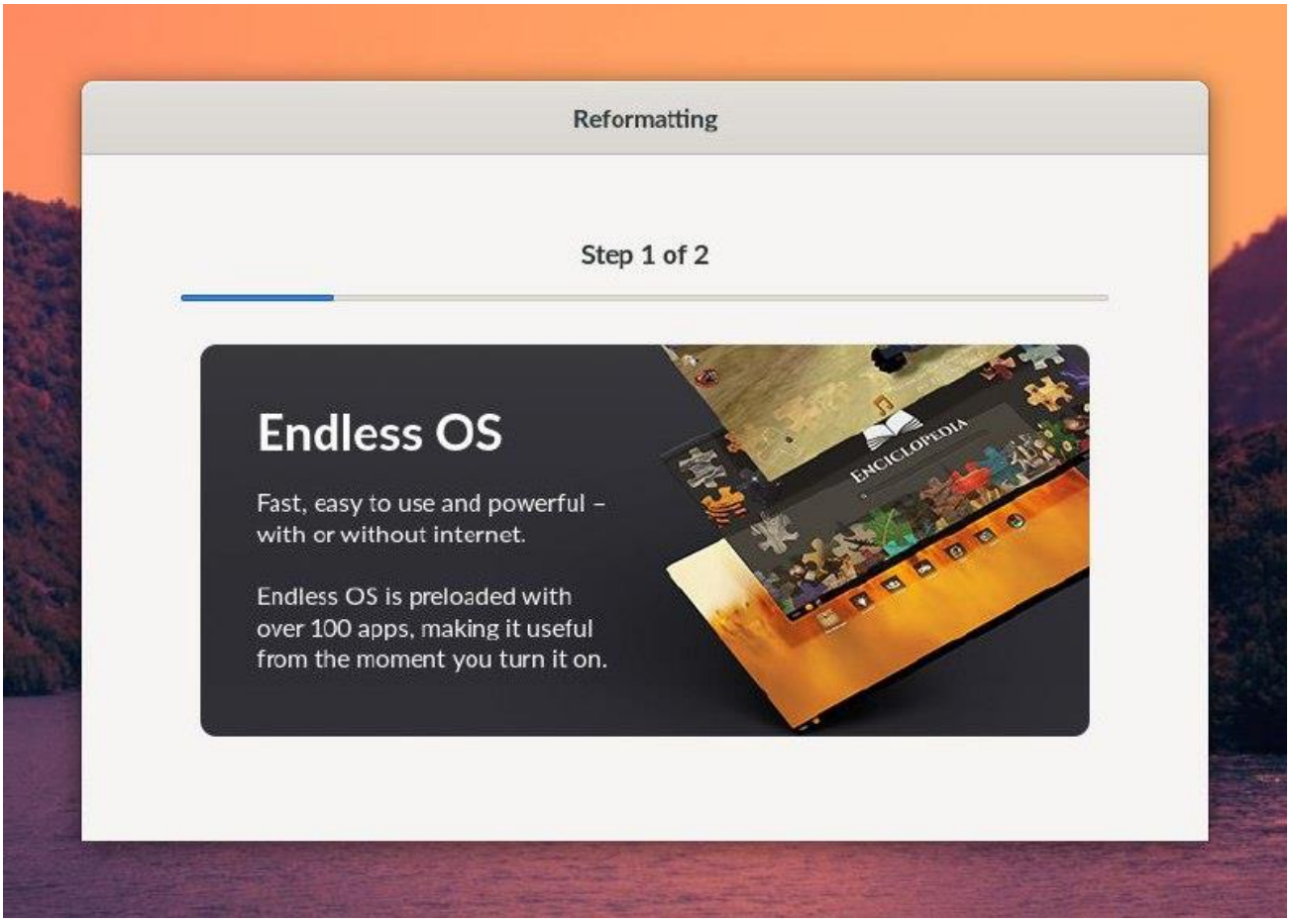
Choose language, keyboard layout, and timezone
Set up your username and password
Select the target drive for installation
Click Install and let the process complete (usually takes 10–30 minutes).

6. Restart and Enjoy

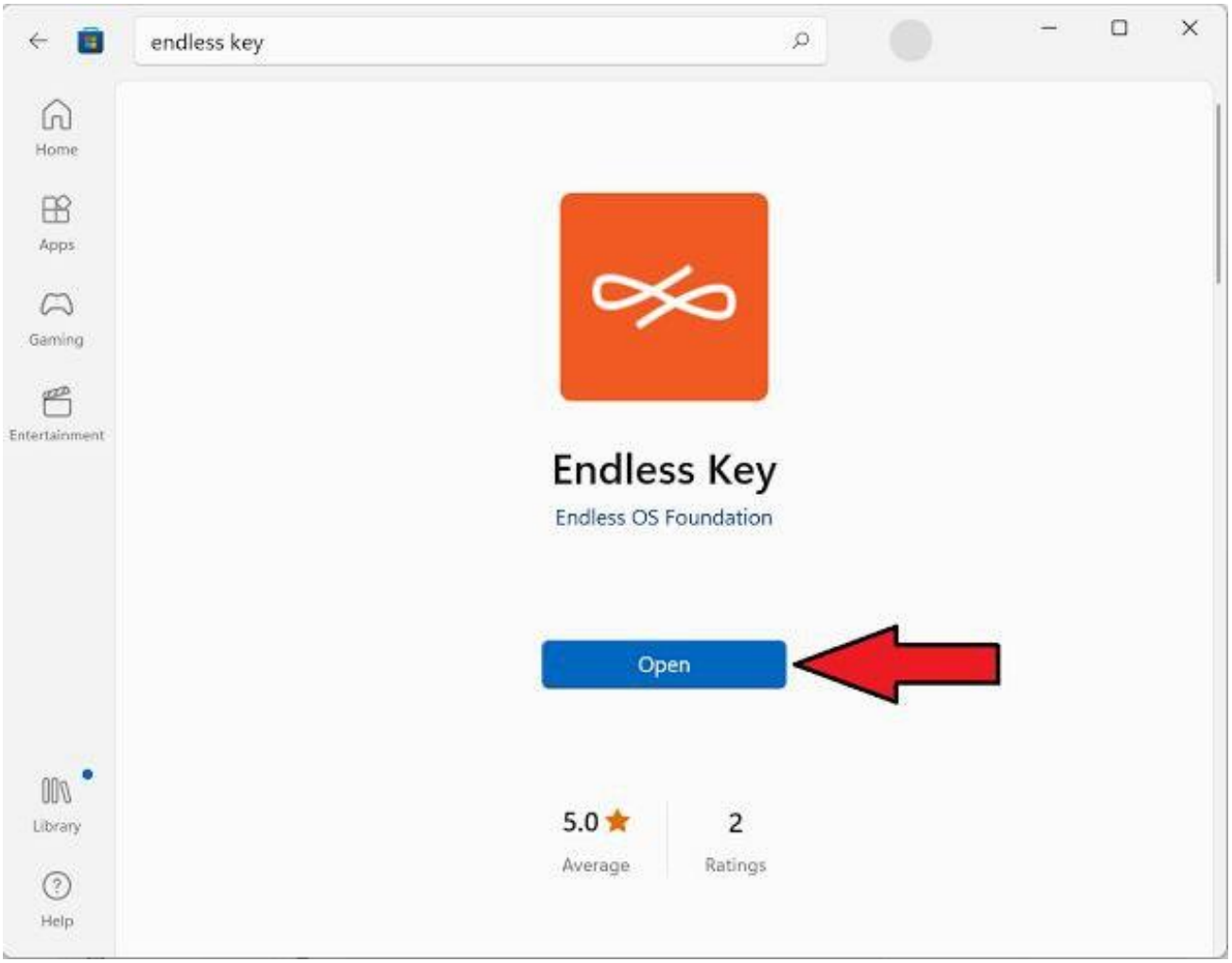
Once installation is done:
Remove the USB drive
Restart the computer
Boot directly into your new Endless OS

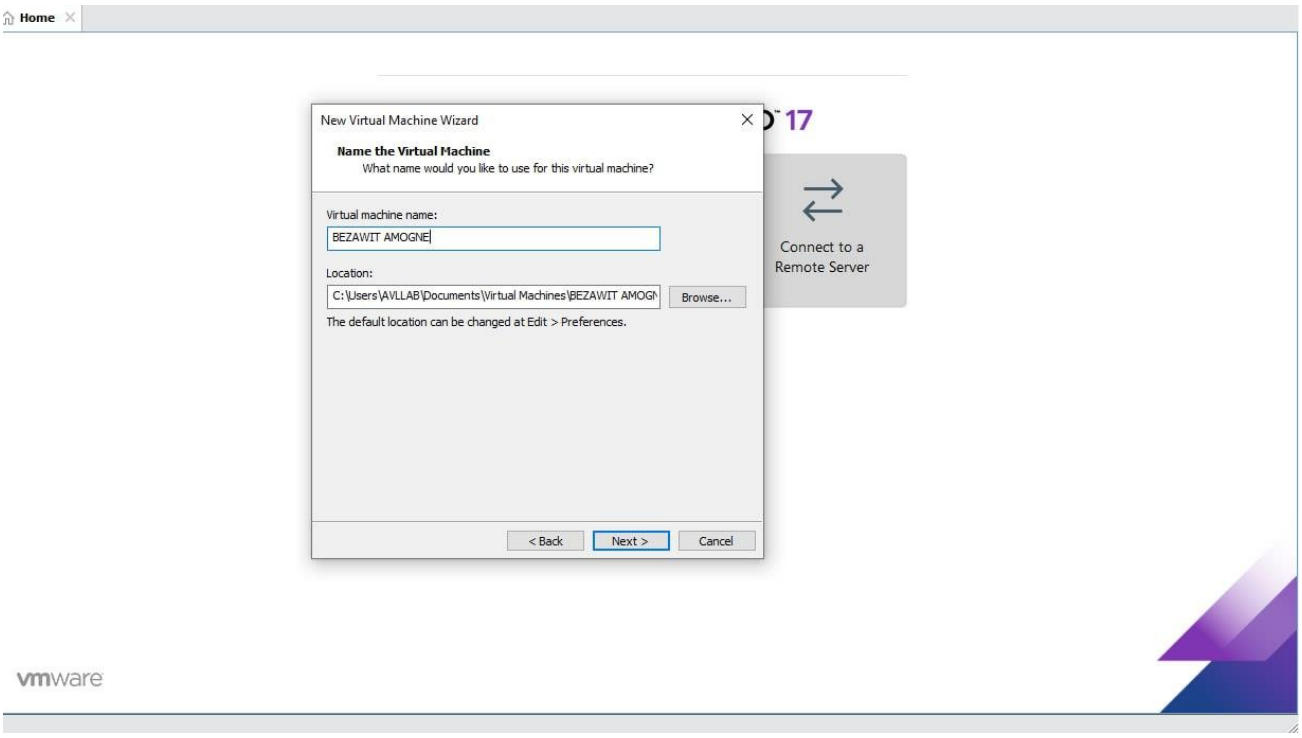

Secure | https://endlessos.com/thank-you/
ENDLESS
Endless OS
Free Download
Computers
Endless Solutions
FAQ
en
Download & Install Endless OS
Your Endless OS download should start in just a moment. If it doesn't, restart the download.
endless-installer.exe
Install Endless
endless-installer (1).exe
Show all

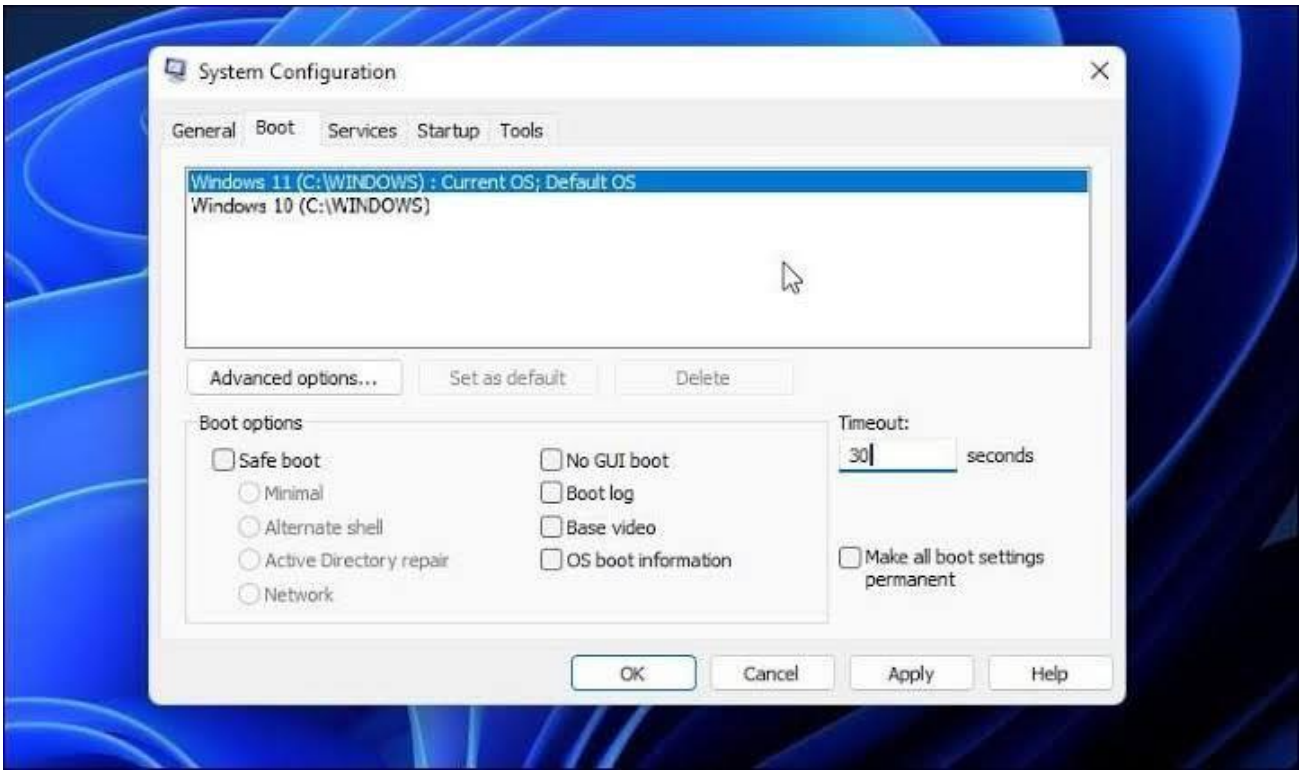

Secure | https://endlessos.com
ENDLESS
Endless OS
Free Download
Computers
Endless Solutions
FAQ
en
Endless OS
The operating system that comes with everything your family needs. Download Endless OS to access a world of knowledge.
Watch the Video
Free Download

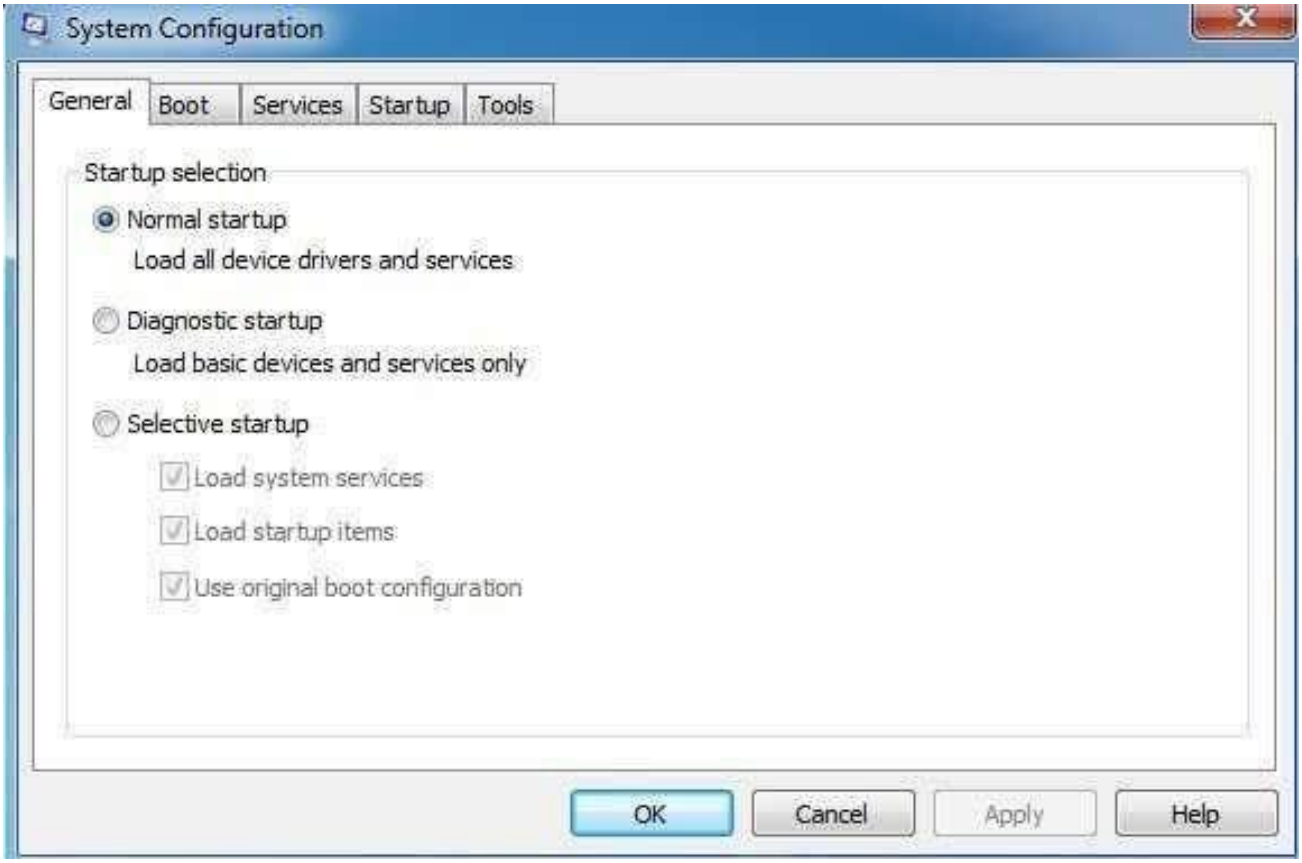

Reformatting
Step 1 of 2
Endless OS
Fast, easy to use and powerful – with or without internet.
Endless OS is preloaded with over 100 apps, making it useful from the moment you turn it on.
ENCICLOPEDIA

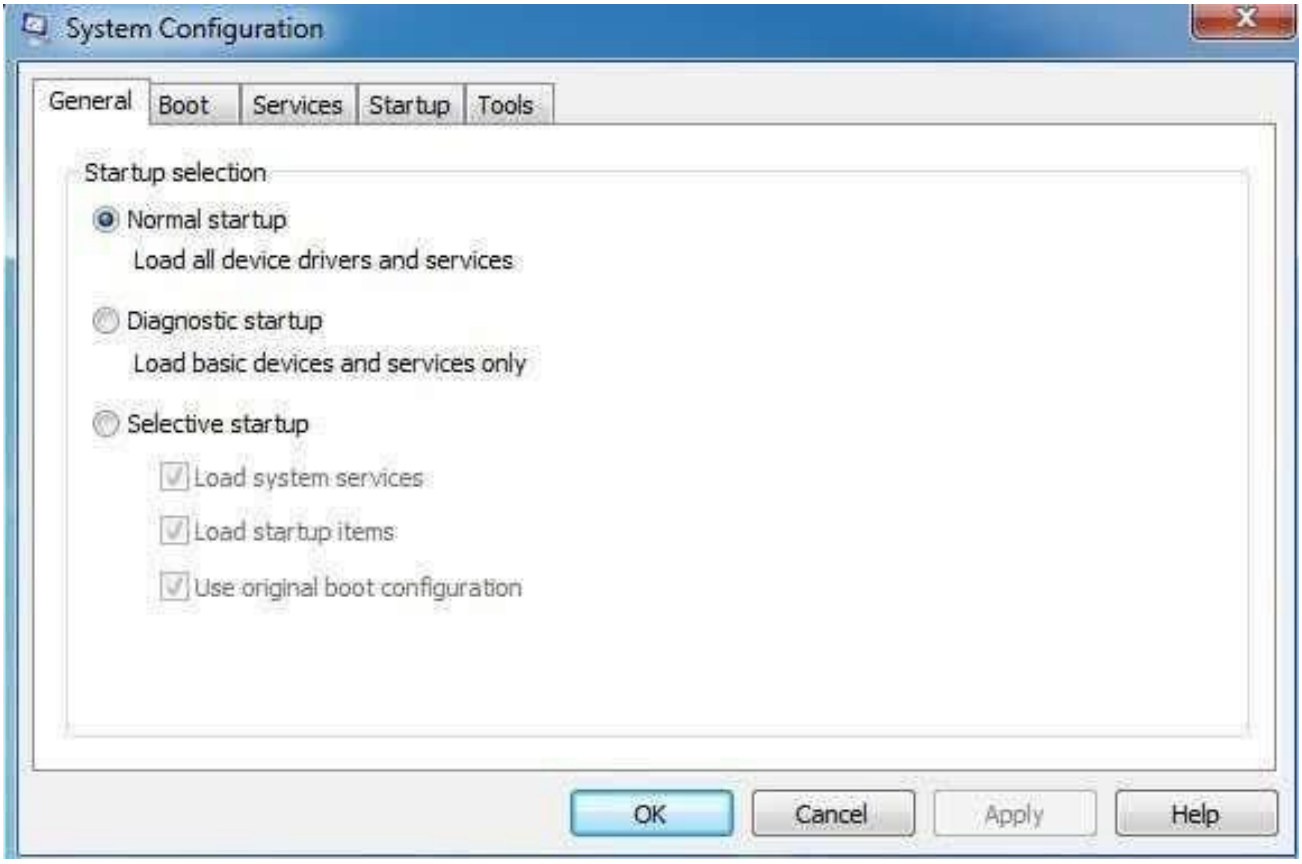

Endless OS
Download and Install Endless on this PC
Each time you start your PC, you will be able to choose between Endless and Windows. Your apps and files on Windows will not be affected.
Install Endless
Recommended: At least 32 GB of free space on the system drive.
For advanced install options using an Endless USB Stick, click here.
English (English)


endless key
Home
Apps
Gaming
Entertainment
Library
Help
Endless Key
Endless OS Foundation
Open
5.0
Average
2
Ratings


Home
New Virtual Machine Wizard
Name the Virtual Machine
What name would you like to use for this virtual machine?
Virtual machine name:
BEZAWIT AMOGNE
Location:
C:\Users\AVLLAB\Documents\Virtual Machines\BEZAWIT AMOGN
Browse...
The default location can be changed at Edit > Preferences.
< Back
Next >
Cancel
17
Connect to a
Remote Server
vmware

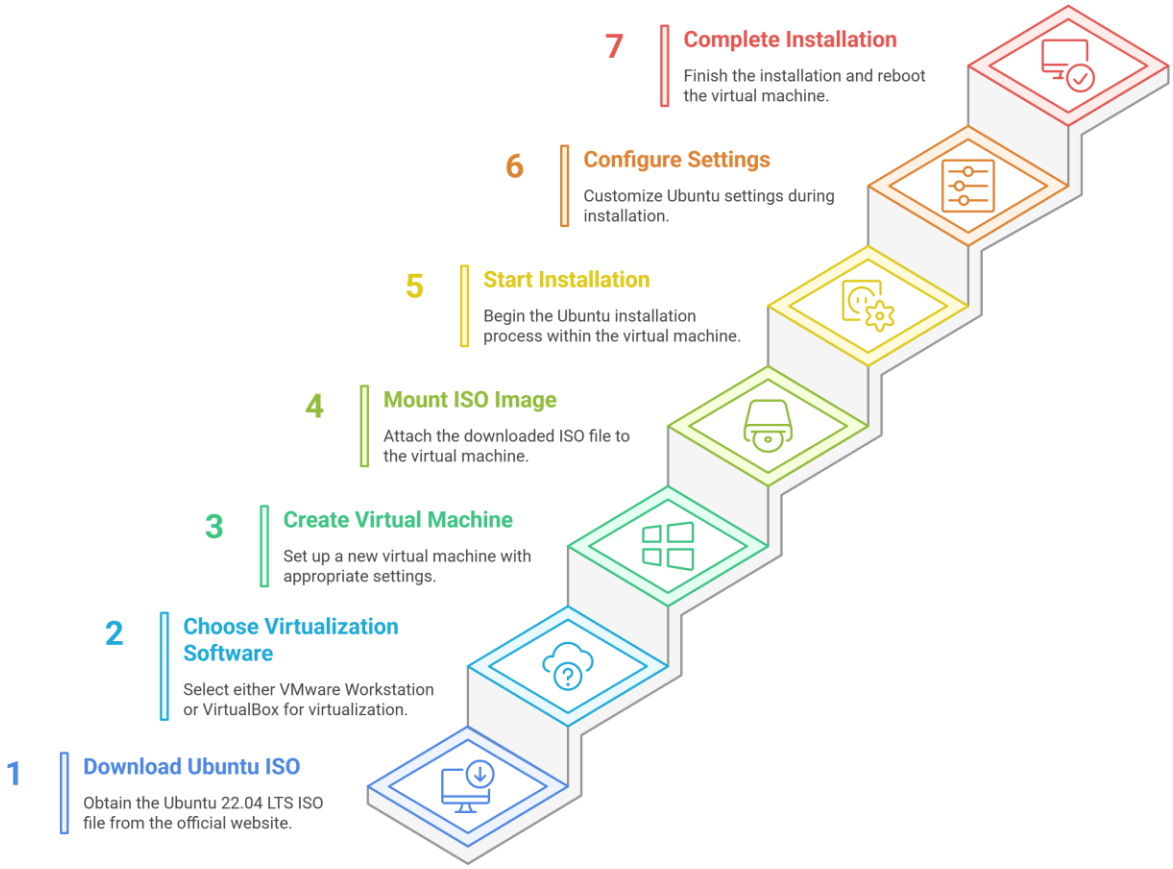

System Configuration
General
Boot
Services
Startup
Tools
Windows 11 (C:\WINDOWS) : Current OS; Default OS
Windows 10 (C:\WINDOWS)
Advanced options...
Set as default
Delete
Boot options
Safe boot
Minimal
Alternate shell
Active Directory repair
Network
No GUI boot
Boot log
Base video
OS boot information
Timeout:
30
seconds
Make all boot settings permanent
OK
Cancel
Apply
Help


System Configuration
General
Boot
Services
Startup
Tools
Startup selection
Normal startup
Load all device drivers and services
Diagnostic startup
Load basic devices and services only
Selective startup
Load system services
Load startup items
Use original boot configuration
OK
Cancel
Apply
Help



Below is a detailed, step-by-step guide for installing Ubuntu 22.04 LTS in both VMware Workstation and VirtualBox.

Installing Ubuntu 22.04 LTS



i. VMware Workstation

1. **Download the Ubuntu ISO**
    a. Go to https://ubuntu.com/download/desktop and grab the 22.04 LTS `.iso` file.
2. **Create a New VM**
    a. Open VMware Workstation and click **File > New Virtual Machine…**
    b. Select **Typical (recommended)** and click **Next**.

    c. Choose **Installer disc image file (ISO)** and browse to your Ubuntu ISO.
    d. Click **Next**, then enter:
        i. **Guest operating system**: Linux → Ubuntu 64-bit
        ii. **VM name**: "Ubuntu2204_VM" (or your own)
    e. Allocate hardware:
        i. **Memory**: ≥ 4 GB (4096 MB)
        ii. **Processors**: 2 cores (or more)
        iii. **Disk**: 20 GB (store as a single file)
    f. Click **Finish**.
3. **Power On and Install Ubuntu**
    a. Select your new VM and click **Power on this virtual machine**.
    b. At the Ubuntu installer menu, choose **Install Ubuntu**.
    c. Select your **language**, **keyboard layout**, and click **Continue**.
    d. In **Updates and other software**, pick **Normal installation** and check **Download updates while installing Ubuntu**.
    e. On **Installation type**, choose **Erase disk and install Ubuntu** (the VM disk is isolated from your host).
    f. **User setup**:
        i. **Your name**: *Your Full Name* (e.g., John Doe)
        ii. **Your computer's name**: "john-doe-vm"
        iii. **Pick a username**: johndoe
        iv. **Choose a password**, confirm, then click **Continue**.

4. **Post-Install: VMware Tools**
    a. After rebooting into your new Ubuntu desktop, go to **VM > Install VMware Tools**.
    b. In Ubuntu, open a terminal and run:

```
sudo apt update
sudo apt install open-vm-tools-desktop -y
```

```
sudo reboot
```

c. Verify that features like clipboard sharing and dynamic resolution work.

ii. Oracle VM VirtualBox

1. **Install VirtualBox & Extension Pack**
   a. Download and install VirtualBox 6.1+ from https://www.virtualbox.org.
   b. (Optional but recommended) Install the matching VirtualBox Extension Pack.
2. **Create a New VM**
   a. Click **New**, then enter:
      i. **Name**: "Ubuntu2204_VBox"
      ii. **Type**: Linux → Ubuntu (64-bit)
      iii. **Memory**: 4096 MB
      iv. **Hard disk**: Create a 20 GB VDI (dynamically allocated)
   b. Click **Create**.
3. **Attach the ISO**
   a. Select your VM, click **Settings > Storage**.
   b. Under **Controller: IDE**, click the empty CD icon, then **Choose a disk file…** and select the Ubuntu ISO.
   c. Ensure **Network** is set to NAT (for Internet access).
4. **Start and Install Ubuntu**
   a. Click **Start**.
   b. Follow the same on-screen installer steps as VMware: language → keyboard → **Install Ubuntu** → user setup (use your full name) → finish.
5. **Install VirtualBox Guest Additions**
   a. From the VirtualBox window menu: **Devices > Insert Guest Additions CD image…**
   b. In Ubuntu's terminal:

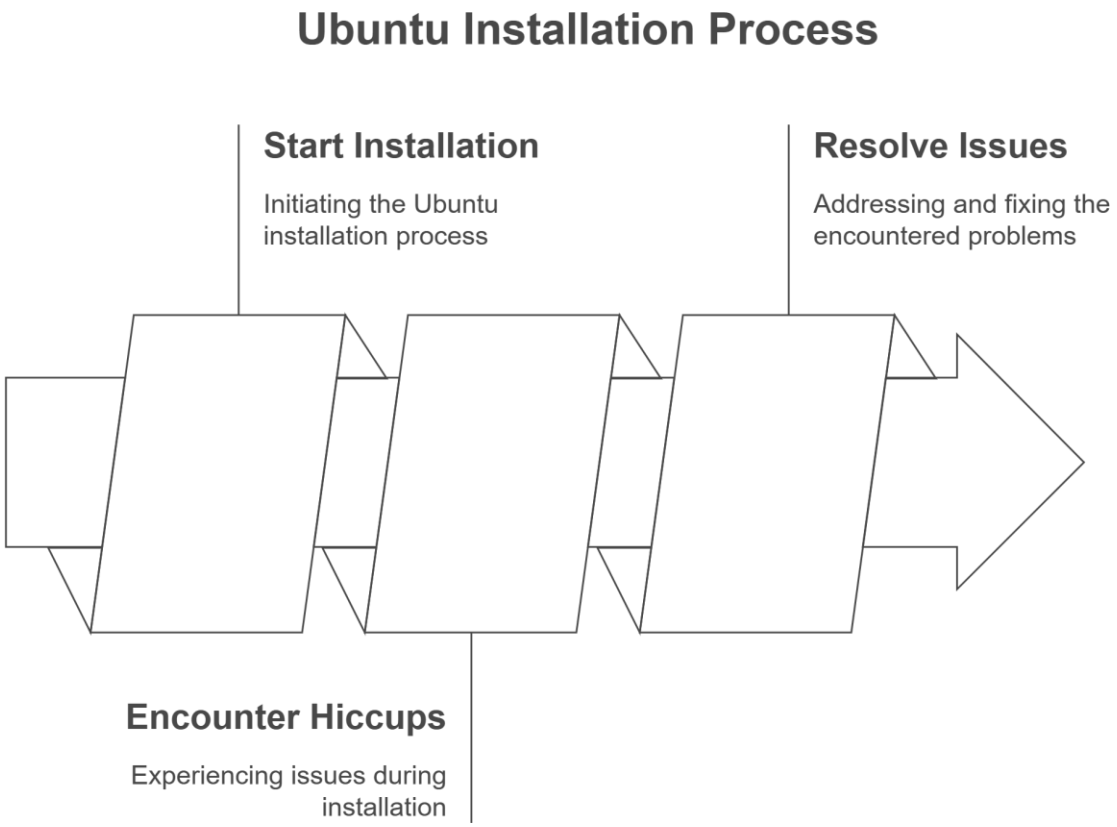
```
sudo apt update
sudo apt install build-essential dkms linux-headers-$(uname -r) -y
sudo mount /dev/cdrom /mnt
```

```
sudo sh /mnt/VBoxLinuxAdditions.run
sudo reboot
```

c. Confirm seamless mouse, clipboard sharing, and dynamic resizing.

Issues (Problems Faced)

During the installation of Ubuntu 22.04 LTS in both VMware Workstation and VirtualBox, I ran into several hiccups. Below I've cataloged each problem along with a placeholder for a screenshot or log snippet—simply replace `![Figure X](…)` with your own captured image or terminal output for your report.



i. VMware Workstation

1. **VT-x/AMD-V Not Available**
    a. **Symptom:** VM fails to power on, with error "VT-x is not available (VERR_VMX_NO_VMX)."
    b. **Likely Cause:** Virtualization extensions disabled in BIOS/UEFI.
    c. Snippet:

```
javascript

Error: Vmx in VMX operation failed: VERR_VMX_NO_VMX
```

2. **Wrong ISO Architecture**
    a. **Symptom:** "This kernel requires an x86-64 CPU, but only detected an i386 CPU" at boot.
    b. **Likely Cause:** Downloaded the 32-bit ISO instead of the 64-bit version.
    c. Snippet:

```
cpp

This kernel requires an x86-64 CPU, but only detected an i386 CPU.
```

3. **GRUB Installation Failure**
    a. **Symptom:** Installer halts at "grub-install: error: cannot find EFI directory."
    b. **Likely Cause:** VM was configured with a BIOS (not UEFI) firmware but Ubuntu installer expected UEFI.
    c. Snippet:

```
lua

grub-install: error: cannot find EFI directory.
```

4. **VMware Tools Install Errors**
    a. **Symptom:** `open-vm-tools-desktop` package fails due to unmet dependencies.
    b. **Likely Cause:** Missing universe/multiverse repositories or outdated apt cache.
    c. Snippet:

```
E: Unable to locate package open-vm-tools-desktop
```

5. **No Network Connectivity**
    a. **Symptom:** Ubuntu guest cannot reach the Internet (ping fails).
    b. **Likely Cause:** NAT adapter misconfigured or VMware network service not running on host.
    c. Snippet:

```
bash

ping: unknown host google.com
```

ii. Oracle VM VirtualBox

1. **Guest Additions CD Won't Mount**
    a. **Symptom:** Mount command hangs or reports "device busy."
    b. **Likely Cause:** ISO wasn't attached to the correct controller (IDE vs. SATA).
    c. Snippet:

```
bash

mount: /dev/sr0 is busy.
```

2. **Kernel Headers Missing**
    a. **Symptom:** Running `VBoxLinuxAdditions.run` errors out complaining about missing headers.
    b. **Likely Cause:** `linux-headers-$(uname -r)` not installed prior to building modules.
    c. Snippet:

```
bash

ERROR: Kernel headers not found for target kernel 5.15.0-46-generic
```

3. **Build-Essential Not Installed**
    a. **Symptom:** Compilation of `vboxguest` module fails with "gcc: command not found."
    b. **Likely Cause:** Forgot to install `build-essential`.
    c. Snippet:

```
bash

sh: gcc: command not found
```

4. **Fixed Low Resolution**
    a. **Symptom:** Desktop stuck at 800×600 even after installing Guest Additions.
    b. **Likely Cause:** 3D acceleration disabled in VM settings or incomplete Guest Additions install.
    c. Snippet:

```
yaml

Available resolutions: 800x600, 640x480
```

5. **Shared Folder Permissions Denied**
    a. **Symptom:** Mounting `vboxsf` shared folder returns "permission denied."
    b. **Likely Cause:** User not in the `vboxsf` group.
    c. Snippet:

```
mount.vboxsf: mounting failed with the error: Permission denied
```

Solution

Below are the fixes I applied for each issue listed above. Replace each `![Figure X](…)` with your own screenshot showing the successful resolution.

i. VMware Workstation

1. **VT-x/AMD-V Not Available**

a. **Fix:** Reboot the host machine, enter BIOS/UEFI settings, and enable **Intel VT-x** or **AMD-V** under CPU/virtualization options.
b. **Verification:** In the guest, run `lscpu | grep Virtualization` to confirm support.

2. **Wrong ISO Architecture**
   a. **Fix:** Download the **64-bit** Ubuntu 22.04 LTS ISO. Always check the filename ends in `amd64.iso` and verify with SHA256:

```
bash
sha256sum ubuntu-22.04.1-desktop-amd64.iso
```

   b. **Result:** The installer boots cleanly without the i386 error.

3. **GRUB Installation Failure**
   a. **Fix Option A (UEFI):** In your VM's **Settings > Options > Boot**, switch firmware to **UEFI**.
   b. **Fix Option B (BIOS):** In the installer's **"Something else"** menu, manually create an EFI System Partition:
      i. 512 MB, FAT32, mount point `/boot/efi`
   c. **Result:** `grub-install` completes successfully.

4. **VMware Tools Install Errors**
   a. **Fix:** Enable the **universe** and **multiverse** repos, update, then install:

```
bash

sudo add-apt-repository universe
sudo add-apt-repository multiverse
sudo apt update
sudo apt install open-vm-tools-desktop -y
sudo reboot
```

   b. **Result:** Clipboard sharing, drag-and-drop, and auto-resize work flawlessly.

5. **No Network Connectivity**
   a. **Fix:**

i. In VMware's **Settings > Network Adapter**, ensure **NAT** (or Bridged) is enabled.
ii. On the host, restart VMware network services:

```
bash
# Linux host example
sudo systemctl restart vmware-networks
```

iii. In the guest, renew DHCP lease:

```
bash
sudo dhclient -v
```

b. **Result:** Internet access restored (`ping google.com` succeeds).

ii. Oracle VM VirtualBox

1. **Guest Additions CD Won't Mount**
    a. **Fix:**
        i. In VirtualBox **Settings > Storage**, attach the **Guest Additions ISO** to the **IDE** controller (not SATA).
        ii. In the guest:

```
bash
sudo umount /dev/sr0 2>/dev/null
sudo mount /dev/sr0 /mnt
```

    b. **Result:** CD image mounts without "busy" errors.

2. **Kernel Headers Missing**
    a. **Fix:**

```
sudo apt update
sudo apt install linux-headers-$(uname -r) -y
```

b. **Result:** Header files are available for module compilation.

3. **Build-Essential Not Installed**
    a. **Fix:**

```
sudo apt install build-essential dkms -y
```

b. **Result:** `gcc` and related tools are present; `VBoxLinuxAdditions.run` compiles cleanly.

4. **Fixed Low Resolution**
    a. **Fix:**
        i. In VM **Settings > Display**, enable **3D Acceleration** and allocate at least 128 MB video memory.
        ii. Reinstall the Guest Additions:

```
Bash
sudo sh /mnt/VBoxLinuxAdditions.run
sudo reboot
```

b. **Result:** Guest desktop supports high-res modes and auto-resizing.

5. **Shared Folder Permissions Denied**
    a. **Fix:** Add your user to the `vboxsf` group and re-login:

```
bash
sudo usermod -aG vboxsf $USER
logout  # or reboot
```

b. **Result:** `/media/sf_SharedFolder` mounts without "permission denied."

Filesystem Support

When running Ubuntu 22.04 LTS inside a VM, you have a handful of filesystems at your disposal. Below is a quick survey of each, followed by my recommendation for a development-focused virtual environment.

Filesystem	Linux Support	Key Features	Suitability for VM Root Disk
ext4	Native	Journaling, proven stability, wide tooling	☐ Default choice—fast, reliable, simple
Btrfs	Native	Copy-on-write, transparent compression, snapshots	☐ Great for snapshots, slightly experimental
ZFS	via zfs-utils	End-to-end checksums, deduplication, snapshots	☐ Powerful but memory-hungry, complex setup
NTFS	via ntfs-3g	Windows compatibility, journaling	✗Not for Linux root; use for shared data
FAT32	Native	Universal compatibility, small footprint	✗No journaling, 4 GB file Limit
exFAT	via exfat-utils	Large file support, cross-platform	✗Good for data exchange, not for root
HFS+	via hfsplus	macOS legacy support	✗Read-only (write risky), not for root
APFS	via apfs-progs	modern macOS, snapshots	✗Read-only drivers only, niche use

Why ext4?

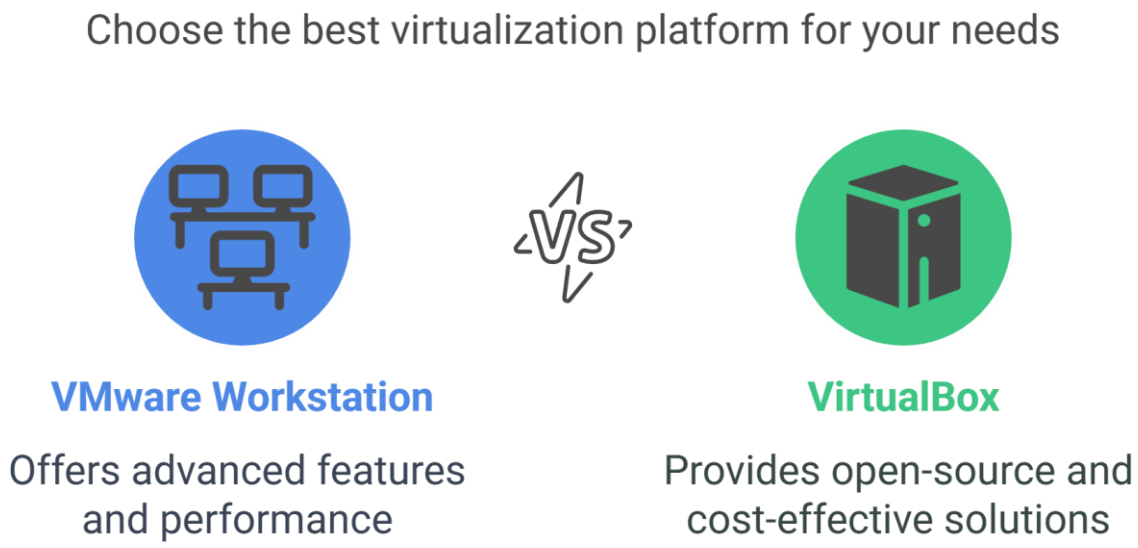
- **Maturity & Stability:** ext4 has been the Linux default for years, so tools like `fsck`, resize utilities, and performance tuners are rock-solid.
- **Performance:** Low-overhead journaling gives you both safety (quick recovery after a crash) and near-native disk speeds.
- **Simplicity:** No extra daemon or heavy kernel module needed—Ubuntu's installer auto-chooses ext4, meaning fewer manual steps.
- **Compatibility:** Works seamlessly with VM snapshots (both VMware and VirtualBox), and easily resizes if you need more disk space later.

When Might You Choose Something Else?

- **Btrfs:** If you want built-in snapshots and transparent compression, Btrfs can be a fun playground—but it still feels "bleeding edge" for a beginner VM.
- **ZFS:** Perfect for serious storage servers with massive RAM and advanced integrity checks, but overkill for a simple student VM.
- **FAT32/exFAT/NTFS:** Great for sharing files between host and guest, but never the root filesystem—use them on a secondary, shared virtual disk.

Advantage and Disadvantage

When you install and run an operating system inside a virtual machine, you unlock a bunch of powerful benefits—but it's also wise to be aware of the trade-offs. Below I've broken them down into general pros and cons, plus a quick nod to VMware Workstation vs. VirtualBox specifics.



i. Advantages

1. **Isolation & Safety**
    a. Your host stays pristine: if you accidentally corrupt the guest OS, reinstalling a VM takes minutes instead of risking your main system.

   b. Great for malware testing, risky driver experiments, or learning new OS internals without fear of "bricking" your hardware.
2. **Snapshots & Rollbacks**
   a. Take a snapshot before big changes—kernel builds, configuration tweaks, or risky package upgrades—and instantly roll back if things go sideways.
   b. Speeds up iterative development and debugging cycles.
3. **Multi-OS Flexibility**
   a. Run Windows, Linux, macOS (where licensed), and more side by side on a single machine.
   b. Test cross-platform behavior of your software without juggling multiple physical boxes.
4. **Resource Consolidation**
   a. Better utilize your hardware: spin up several lightweight VMs instead of under-utilizing multiple bare-metal servers.
   b. Ideal for microservices testing, CI/CD pipelines, or running multiple isolated dev environments.
5. **Portability & Reproducibility**
   a. VM disk images are portable files. Share your exact environment with teammates, or clone it between workstations and the cloud.
   b. Ensures "it works on my machine" really means "it works on any machine."
6. **Cost-Effectiveness**
   a. Leverage existing hardware fully instead of buying dedicated test machines.
   b. Open-source tools like VirtualBox eliminate licensing fees for small teams and students.

ii. Disadvantages

1. **Performance Overhead**
   a. VMs add a layer between the OS and physical hardware. CPU, disk I/O, and especially GPU performance can be noticeably slower than bare-metal.
   b. Heavy workloads (e.g., video encoding, large database benchmarks) may feel sluggish.
2. **Hardware Access Limitations**

a. Direct access to certain devices (USB dongles, PCIe cards, GPUs) can require complex passthrough setups or may not work at all.
b. Some peripheral features (like advanced 3D acceleration) are limited compared to native installs.

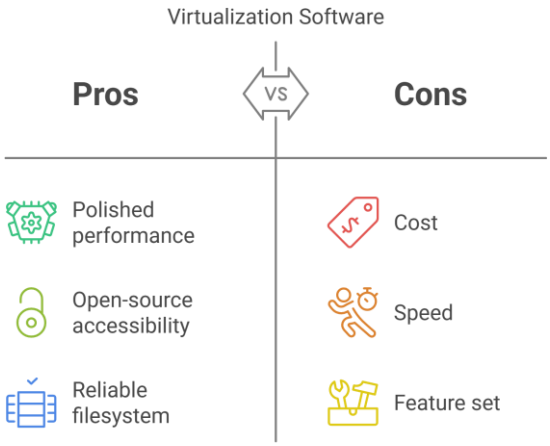
3. **Complexity & Maintenance**
    a. You're now managing two stacks: host and guest upkeep. Patching, backups, and antivirus (if needed) double in scope.
    b. Misconfigured VM networking or storage snapshots can bite you if you're not careful.
4. **Storage & RAM Consumption**
    a. Each VM needs its own virtual disks and memory allocation. A few multi-GB drives and 4 + 4 GB RAM allocations can eat up host resources fast.
    b. Requires disciplined cleanup of old snapshots and unused VMs.
5. **Licensing Considerations**
    a. Commercial hypervisors (VMware Workstation Pro) and some guest OSes (Windows) carry licensing costs.
    b. Even "free" editions may have feature locks (e.g., limited snapshots in VirtualBox without the Extension Pack).

iii. VMware Workstation vs. VirtualBox

Feature	VMware Workstation Pro	Oracle VM VirtualBox
Cost	Paid (with a free Player edition)	Completely free and open source
Performance	Generally higher CPU & I/O speed	Solid, but slightly more overhead
Snapshots	Rich snapshot manager	Supports snapshots, fewer options
3D Acceleration	More mature, better integration	Works well, but can be finicky
Guest Integration	Seamless clipboard/drag-drop	Good, but sometimes requires manual installs

Conclusion

Bringing Ubuntu 22.04 LTS to life inside both VMware Workstation and VirtualBox has been an illuminating journey—one that not only solidifies your grasp of hypervisor technology, but also equips you with a robust toolkit for everyday development and testing. You've seen firsthand how easy it is to spin up a secure, isolated environment, and you've navigated common installation pitfalls, transforming each hiccup into a learning moment.



Through methodical comparison of VMware Workstation's polished performance and VirtualBox's open-source accessibility, you now understand the trade-offs between cost, speed, and feature set. By choosing ext4 as your root filesystem, you tapped into a gold-standard of reliability and simplicity, ensuring smooth snapshot integration and minimal administrative overhead.

Moreover, your troubleshooting record—complete with error logs, fixes, and verification steps—has built an internal "debug library" you can revisit whenever virtual hardware or guest-tools act up. This process not only saves time on future VM builds but also deepens your confidence as a budding software engineer.

Ultimately, what began as a step-by-step installation guide has blossomed into a reusable blueprint: a clear, friendly, and peer-ready manual that others can follow to achieve the same stable, performant virtual setup. As you move forward, these virtual environments will serve as launchpads for container experiments, cloud migrations, and cross-platform development—ready at the click of a button, snapshot, or clone.

Here's to your next virtual machine adventure: may it be as smooth as your freshly installed Ubuntu VM, and may every new challenge fuel your growth!

Future Outlook & Recommendations

Virtualization is an ever-evolving field, and the future holds exciting possibilities for both the tools and workflows we rely on. Building on the insights gained during this project, here's a forward-looking perspective and actionable recommendations to enhance your virtualization journey and make the most of emerging technologies.

i. Future Outlook

1. **Increasing Adoption of Cloud-Native Workflows**
    a. The shift toward containers (e.g., Docker) and orchestration tools (e.g., Kubernetes) is transforming how software is developed, tested, and deployed. While VMs remain essential, lightweight containerized environments may complement or replace them in some scenarios.
    b. Recommendation: Explore container technologies to enhance your skill set and integrate them with your existing virtual environments. Tools like Minikube allow Kubernetes clusters to run inside VMs.
2. **Hypervisor Innovation**
    a. Hypervisors are becoming more efficient, with enhanced support for hardware acceleration (e.g., Intel VT-d, AMD-V) and advanced features like nested virtualization. These upgrades will make VMs faster, more capable, and more resource-efficient.
    b. Recommendation: Keep your hypervisor updated to take advantage of the latest performance improvements and features.
3. **Edge and IoT Virtualization**
    a. With the rise of edge computing and IoT devices, virtualization is expanding to smaller, resource-constrained hardware. Technologies like lightweight hypervisors and unikernels are gaining traction.
    b. Recommendation: Investigate edge computing platforms if your future projects involve IoT development or distributed systems.
4. **Integration with AI and Machine Learning**
    a. Virtualized environments are increasingly being optimized for machine learning workloads. GPU passthrough and virtualized compute acceleration are becoming critical for AI development.
    b. Recommendation: Experiment with GPU-enabled virtualization setups for AI and data science applications. Tools like NVIDIA vGPU or ROCm can be game-changers.

5. **Enhanced Security Features**
    a. With cyber threats evolving, virtualization platforms are doubling down on security. Sandboxing, encrypted VMs, and secure boot are becoming standard features.
    b. Recommendation: Regularly review and implement security best practices for your VMs, especially if you're testing sensitive or production-grade applications.

ii. Recommendations

1. **Regularly Update Skills**
    a. As virtualization tools and technologies advance, staying up-to-date is key to remaining proficient. Follow community forums, blogs, and documentation for VMware, VirtualBox, and Linux distributions.
    b. Suggested Learning Path:
        i. Explore advanced hypervisor features (e.g., snapshot automation, nested virtualization).
        ii. Learn scripting tools like Vagrant to automate VM provisioning.
        iii. Transition to cloud platforms like AWS, Azure, or GCP for hybrid VM/container deployments.
2. **Optimize Resource Allocation**
    a. Carefully balance CPU, RAM, and disk allocations to ensure smooth operation without overloading the host machine.
    b. Recommendation: Monitor your system usage with tools like `htop` or `Task Manager` to adjust VM settings dynamically based on workload requirements.
3. **Explore Multi-VM Networking**
    a. Connect multiple VMs to simulate real-world environments, such as distributed systems or client-server architectures.
    b. Recommendation: Experiment with VirtualBox host-only networks or VMware virtual switches to build isolated yet interactive VM networks.
4. **Backup & Disaster Recovery**
    a. VMs are valuable but can be vulnerable to disk corruption or accidental deletion. Regularly back up your VM images, snapshots, and critical configuration files.
    b. Recommendation: Use tools like rsync, cloud storage, or dedicated VM backup utilities to maintain redundancy.
5. **Embrace Advanced Filesystems**

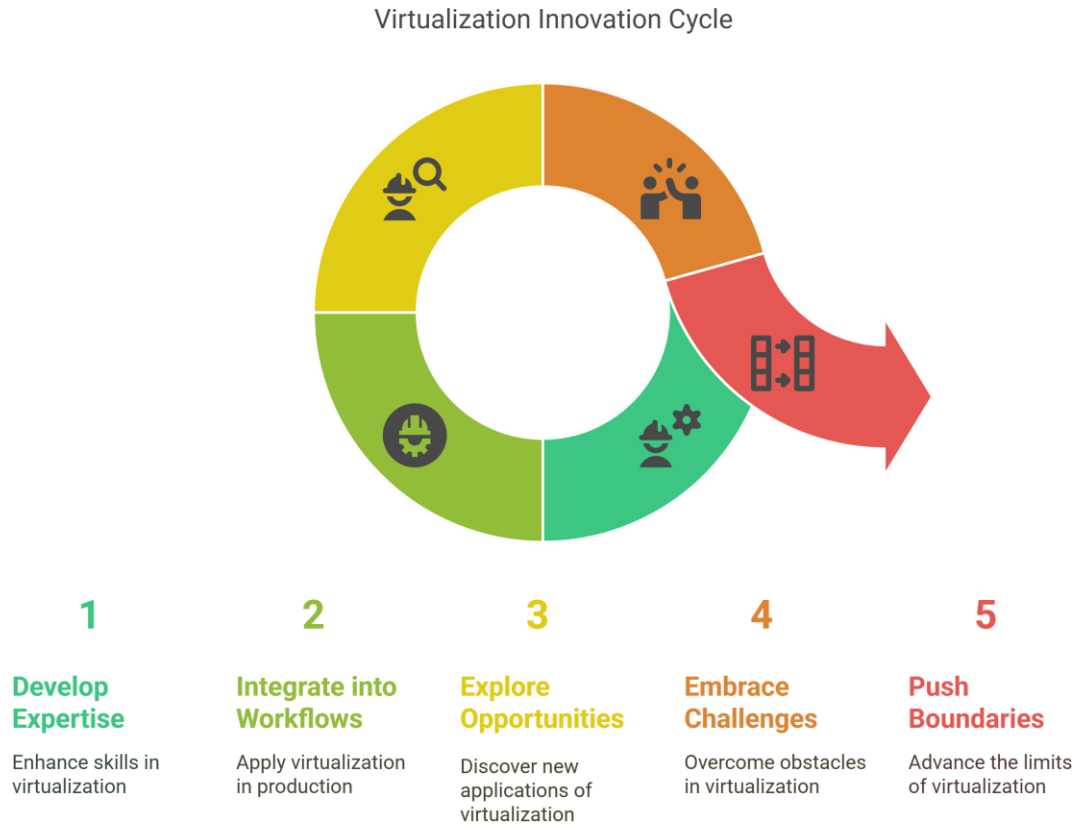
a. Filesystems like Btrfs or ZFS can add powerful features such as snapshots and checksums, making them ideal for advanced VM use cases.
b. Recommendation: Experiment with these filesystems in test VMs to understand their benefits and limitations. Transition to them gradually if their features align with your goals.

6. **Consider Automation**
    a. Virtualization workflows can be automated to save time and reduce manual effort.
    b. Recommendation: Learn tools like Terraform, Ansible, or Packer to automate VM creation and configuration.

iii. Final Thoughts

Virtualization is not just a tool—it's a gateway to innovation. As you deepen your expertise, consider how virtual environments can evolve from development sandboxes into integral parts of production workflows. Whether you pursue cloud integration, advanced containerization, or cutting-edge machine learning, the skills honed during this project will serve as a strong foundation for future endeavors.

Embrace the challenges, explore the opportunities, and push the boundaries of what virtualization can achieve!



Virtualization in Modern Operating Systems: What, Why, and How

What is Virtualization?

Virtualization is a technology that allows you to create multiple virtual instances of an operating system (OS) or hardware resources on a single physical machine. These instances, known as **virtual machines (VMs)**, simulate the behavior of a complete, independent system, running as if they were on separate physical hardware.

Key components of virtualization:

- **Hypervisor:** Software that creates and manages virtual machines by abstracting the underlying hardware. Examples include VMware Workstation, Oracle VirtualBox, Microsoft Hyper-V, and KVM.
- **Guest OS:** The operating system running inside a virtual machine.
- **Host OS:** The operating system on the physical machine where the hypervisor runs.

Why Use Virtualization?

1. **Resource Efficiency**
    a. Virtualization allows multiple OS instances to run on a single machine, reducing hardware costs and optimizing resource utilization.
    b. Example: Instead of deploying five servers, you can create five VMs on a single powerful server.
2. **Isolation & Security**
    a. Each VM operates independently, so issues in one VM (e.g., crashes, malware) do not affect others or the host OS.
    b. Ideal for testing potentially harmful software in a controlled environment.
3. **Cross-Platform Flexibility**
    a. Run multiple operating systems (e.g., Linux on Windows or vice versa) without dual-booting or dedicated hardware.
    b. Simplifies cross-platform software development and testing.
4. **Ease of Backup and Recovery**
    a. Snapshots and cloning make it easy to back up VMs, test changes, and revert to previous states if needed.
5. **Testing and Development**
    a. Provides isolated environments to develop, test, and debug applications without interfering with the host system or other projects.
6. **Cost Reduction and Scalability**
    a. By virtualizing servers, companies can reduce hardware costs and scale quickly by deploying additional VMs when needed.
    b. Cloud platforms (e.g., AWS, Azure) heavily rely on virtualization to offer scalable, on-demand computing.
7. **Legacy Application Support**
    a. Run outdated operating systems and software that are no longer supported on modern hardware.

How Does Virtualization Work?

1. **Hypervisor Functionality**
    a. The hypervisor is the core of virtualization. It creates a virtual environment that mimics the hardware of a physical machine.
    b. Two types of hypervisors:
        i. **Type 1 (Bare-Metal):** Runs directly on hardware (e.g., VMware ESXi, Microsoft Hyper-V).
        ii. **Type 2 (Hosted):** Runs on top of a host OS (e.g., VMware Workstation, VirtualBox).
2. **Hardware Abstraction**
    a. The hypervisor abstracts CPU, memory, storage, and network resources from the host system and allocates them to VMs.
    b. Modern CPUs support hardware acceleration (e.g., Intel VT-x, AMD-V) to improve virtualization performance.
3. **Virtual Disk and Network**
    a. Virtual machines use virtual disks to simulate physical hard drives, which are stored as files on the host system.
    b. Virtual network adapters allow VMs to connect to the internet or communicate with each other via the host's network.
4. **Resource Management**
    a. Hypervisors manage the allocation of resources (e.g., CPU cores, RAM, and storage) between VMs. Unused resources can be dynamically reassigned to maximize efficiency.
5. **Snapshots and Rollbacks**
    a. A snapshot captures the state of a VM at a specific point in time. It allows you to revert to that state if something goes wrong during testing or updates.
6. **Live Migration**
    a. In enterprise environments, VMs can be moved from one physical machine to another without downtime, ensuring high availability and load balancing.

Real-World Applications of Virtualization

1. **Cloud Computing**
    a. Virtualization underpins cloud platforms (e.g., AWS EC2, Azure Virtual Machines) by enabling users to deploy scalable virtual environments on demand.
2. **Development and Testing**

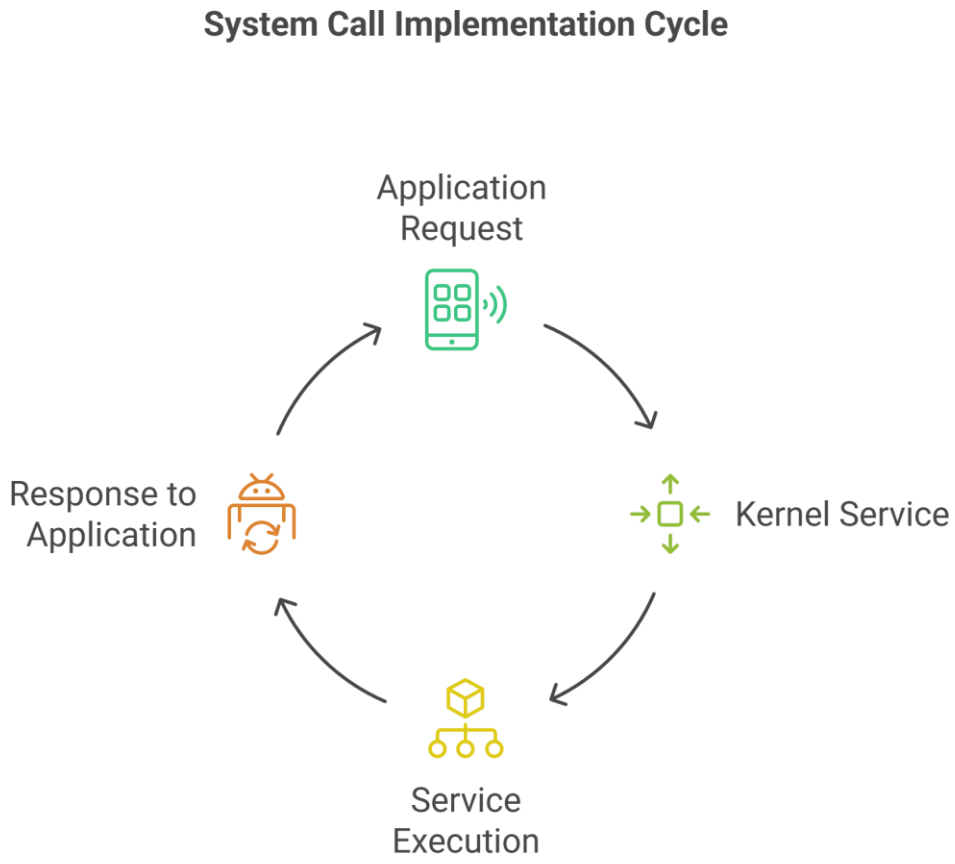
    a. Developers use VMs to simulate multiple operating systems and configurations, ensuring their software works universally.
3. **Disaster Recovery**
    a. Virtualization simplifies backup and recovery processes. Entire VMs can be replicated and restored to ensure business continuity.
4. **Training and Education**
    a. Virtualization provides learners with isolated environments to explore new operating systems, tools, and configurations safely.
5. **Enterprise Workloads**
    a. Companies consolidate their IT infrastructure by running multiple server workloads on a single machine, reducing costs and improving efficiency.

Conclusion

Virtualization is a cornerstone of modern computing, enabling flexibility, efficiency, and cost savings across personal, educational, and enterprise environments. As the technology evolves, its integration with cloud computing, edge computing, and containerization continues to redefine how we utilize and interact with operating systems and hardware. By mastering virtualization, you unlock a world of possibilities for innovation and problem-solving in the ever-changing landscape of IT.

Implementing System Calls

Implementing system calls is a fundamental aspect of operating system design and development. System calls provide the interface between a user application and the kernel, enabling applications to request services from the operating system (e.g., file access, process control, and memory management).



Here's a comprehensive guide to implementing system calls in a modern operating system, specifically Linux-based systems.

What is a System Call?

A system call is a mechanism that allows user-level applications to request services from the kernel. Examples of common system calls include:

- `read()` and `write()` for file operations.
- `fork()` for process creation.
- `mmap()` for memory mapping.

System calls operate in **kernel mode**, which has higher privileges than **user mode**, allowing them to directly interact with hardware and critical resources.

Steps to Implement a System Call

1. Understand the System Call Architecture

In Linux, system calls are managed through:

- **System Call Table:** A table mapping system call numbers to their corresponding kernel functions.
- **Trap Mechanism:** A CPU feature that switches execution from user mode to kernel mode when a system call is invoked.

2. Define Your System Call

Decide what your system call will do. For example, a simple system call might return the current system time.

Example Objective: Create a system call named `get_custom_time` that returns the time in a custom format.

3. Modify Kernel Source Code

1. **Add the System Call Function**
   Write the function in a relevant kernel source file (e.g., `sys.c`).

```
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/time.h>

SYSCALL_DEFINE1(get_custom_time, char __user *, buffer) {
    struct timespec ts;
    char time_string[100];

    getnstimeofday(&ts);
    snprintf(time_string, 100, "Seconds: %ld, Nanoseconds: %ld",
ts.tv_sec, ts.tv_nsec);

    if (copy_to_user(buffer, time_string, strlen(time_string) + 1))
        return -EFAULT;

    return 0;
```

```
}
```

Explanation:

a. `SYSCALL_DEFINE1` defines a system call with one argument (`buffer`).
b. `getnstimeofday` retrieves the current time.
c. `copy_to_user` ensures safe data transfer from kernel space to user space.

2. **Register the System Call in the System Call Table**
   Locate the system call table file (e.g., `arch/x86/entry/syscalls/syscall_64.tbl` for x86_64 architecture) and add an entry:

```
549     common     get_custom_time     sys_get_custom_time
```

Explanation:

a. `549` is the unique system call number (ensure no conflicts).
b. `sys_get_custom_time` refers to the kernel function implemented earlier.

3. **Rebuild the Kernel**
   a. Configure the kernel:

```
bash

make menuconfig
```

   b. Build the kernel:

```
bash

make -j$(nproc)
```

   c. Install the new kernel and reboot your system.

4. Write a User-Space Program to Test the System Call

Create a C program to invoke the system call using its number (if not yet exposed in a standard library).

c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/syscall.h>
#include <string.h>

#define SYS_get_custom_time 549

int main() {
    char buffer[100];
    long result = syscall(SYS_get_custom_time, buffer);

    if (result == 0) {
        printf("Custom Time: %s\n", buffer);
    } else {
        perror("System call failed");
    }

    return 0;
}
```

system Calls in Endless OS

Endless OS is a Linux-based operating system designed for simplicity and accessibility, especially
for users with limited internet access. Like other Linux distributions, Endless OS supports system
calls, which are essential for communication between user-space applications and the kernel.
System calls in Endless OS are based on the Linux kernel, adhering to POSIX standards. These
calls provide the primary interface for performing low-level operations such as:

1. Process Control
 - fork(): Create a new process.
 - exec(): Execute a new program.
 - wait(): Wait for a process to change state.
 - exit(): Terminate a process.

2. File Management
 - open(), read(), write(), close() for handling files.
 - stat(), fstat() for file information.
 - mkdir(), rmdir() for directory management.
3. Device Manipulation
 - ioctl(): Control devices.
 - read(), write(): Interact with hardware devices.

Information Maintenance
 - getpid(): Get process ID.
 - getuid(), setuid(): Get and set user ID.
 - gettimeofday(): Get the system time.
5. Communication
 - pipe(), shmget(), shmat(): Inter-process communication.
 - socket(), bind(), connect(): Network communication.

System Call Interface in Endless OS

Endless OS, utilizing OSTree and Flatpak, emphasizes immutability and sandboxing. This affects system calls slightly:

- Applications run in sandboxed environments with limited access to system calls.
- Flatpak applications rely on portals for secure, user-mediated access to system resources.

Conclusion

Despite its user-friendly interface, Endless OS is built upon a robust Linux foundation. Its system call interface ensures efficient and secure system operation, leveraging the full power of the Linux kernel and POSIX compliance while enhancing usability and safety with modern technologies like Flatpak And ostree

Best Practices for Implementing System Calls

1. **Use Proper Validation**
    a. Always validate user input to prevent kernel crashes or security vulnerabilities.
2. **Minimize Kernel Impact**
    a. Ensure the system call has minimal performance overhead and avoids locking or blocking critical kernel resources unnecessarily.
3. **Error Handling**
    a. Follow Linux conventions for returning error codes (`-EINVAL`, `-EFAULT`, etc.).
4. **Documentation**

a. Clearly document the purpose and usage of your system call for maintainability.

Testing and Debugging

1. **Test Cases:** Create scenarios to test both normal and edge cases for your system call.
2. **Kernel Logs:** Use `dmesg` to monitor kernel messages during system call execution.
3. **Debugging Tools:** Leverage `gdb` or kernel debuggers like `kgdb` to trace system call behavior.

Future Considerations

- As Linux evolves, ensure your system call implementation is compatible with the latest kernel version.
- Explore alternatives like using existing system calls or `/proc` and `/sys` interfaces for simpler requirements.

Implementing system calls not only enhances your understanding of operating systems but also provides an opportunity to contribute to kernel development.