



Bahir Dar University Bahir Dar Institute of Technology

Faculty of Computing

Department of Software Engineering

Course Title: Operating System and System Programming

Documentation for Mac-OS Sonoma

Name: Robel Yihelu

ID: BDU1602340

Section: B



Submitted to: Lec. Wendmu B.

April, 2017

CONTENT

Contents

Introduction	1
OBJECTIVES	2
REQUIREMENTS	4
I. Hardware	4
1. System Specifications	4
II. Software.....	4
1. Core Software	4
2. Additional Requirements.....	5
Installation steps	6
Issues (problems) with their solutions	20
File System Support in macOS Sonoma (Version 14).....	22
Main Advantages of macOS Sonoma	25
Disadvantages of macOS Sonoma	26
Conclusion.....	27
Future Outlook & Recommendations.....	28
Virtualization in macOS	30
What is Virtualization?	30
Why is Virtualization Critical?	31
How Does Virtualization Work?.....	32
Virtualization in macOS Sonoma – Particular Contemplations	34
Implementing system call.....	35
What Is a System Call?.....	35

Introduction

macOS Sonoma (version 14), Apple's newest version of the desktop operating system, was announced at the 2023 Worldwide Developers Conference (WWDC) and represents a substantial leap forward in the evolution of Apple's UNIX-based operating system. With new programming and displays for the computing environment, Sonoma brings additional performance optimizations, additional security and usability aspects with it, making it a versatile operating system use by just about anyone, not just a creative. Sonoma is also built on the UNIX space with UNIX 03 compliance, and is also built on a Darwin (XNU) kernel (the kernel is a hybrid using both Mach and BSD) that also means that it meets the requirements for enterprise-grade stability, security, and interoperability with UNIX, including support of POSIX.

Built to take advantage of the architectural efficiency of Apple Silicon (M1/M2 chips), Sonoma is designed to maximize the utility of that efficiency to maximise hardware and software performance, energy savings for battery-powered devices, compatibility with newer GPU architecture to provide native support for expanding workflows based on advanced machine learning methods. A few of the unused highlights of Sonoma are intuitively widgets on the desktop surface and within the fast see, amusement mode to improve execution for less ping times or idleness, video conferencing choices with moderators overlay alternatives, and an programmed workspace supervisor called Arrange Supervisor. Sonoma also has enterprise-grade security features, which include: hardened sandboxing, APFS (Apple File System) on encrypted volumes, and on-device computing imitates for Siri and related workflows.

This study undertakes a comprehensive analysis of macOS Sonoma's standardization, deployment methodologies, and configuration protocols, with a focus on virtualization challenges arising from Apple's proprietary hardware constraints. By assessing establishment workflows and compatibility impediments in non-Apple virtualized situations (e.g. VMware), the project provides a critical examination of Sonoma's adaptability in heterogeneous IT infrastructures. Additionally, the research explores the OS's alignment with contemporary DevOps practices, including containerization and CI/CD pipeline integration, while addressing the implications of Apple's closed ecosystem on cross-platform development. Through experimental assessment, this work points to set up a methodological system for optimizing macOS Sonoma in scholastic, venture, and designer settings, bridging hypothetical UNIX standards with viable usage challenges.

OBJECTIVES

1. **Install macOS Sonoma on virtual machines like VMware even if it's a bit tricky on non-Apple systems.**

This objective center on overcoming Apple's equipment limitations to effectively introduce and run macOS Sonoma on virtualization stages like VMware. It includes utilizing tools like VMware Unlocker, designing virtual equipment, and tending to compatibility issues to mimic macOS situations on non-Apple machines.

2. **Set up user accounts**

Setting up nearby client accounts post-installation makes a difference oversee framework benefits, personalize situations, and keep up secure get to control for different clients amid testing or multi-user utilize cases in a virtualized macOS Sonoma framework.

3. **Explore supported filesystems like APFS, exFAT, and HFS+—and understand why Apple prefers APFS for modern macOS systems.**

This incorporates distinguishing and testing different macOS-compatible record frameworks, with a essential center on APFS. You'll examine its advantages such as snapshotting, encryption, and SSD optimization, as well as compare it to legacy formats like HFS+ and cross-platform formats like exFAT.

4. **Fix installation and usage issues, document the problems you face, and explain how you solved them for future users.**

This objective promotes hands-on troubleshooting during virtualization, including issues like boot loops, network errors, or hardware compatibility. The goal is to create a technical reference to help future users avoid or resolve similar issues efficiently.

5. **Understand virtualization—what it is, why it's useful, and how it works specifically on macOS Sonoma using tools like VMware.**

This involves studying the concepts and mechanisms behind virtualization, including hypervisors, guest OS handling, and how Sonoma leverages or challenges these when deployed virtually. Special attention is given to Apple Silicon and x86 virtualization methods.

6. **Write shell scripts to automate basic system tasks and improve file security and system performance on macOS.**

The goal here is to implement shell scripting (e.g., in zsh) to automate user and system-level operations—like backups, system updates, or log cleanup—improving usability, performance, and security in day-to-day workflows on Sonoma.

7. **Evaluate how macOS supports cross-platform development—especially for people using both Apple and non-Apple systems.**

This objective analyzes how developers can use macOS alongside other OSes in hybrid environments, focusing on compatibility with languages, tools (like Docker, Git), and APIs. It also reviews the limitations imposed by Apple’s closed ecosystem.

8. **Develop small, practical UNIX-based scripts in the macOS shell environment to improve daily workflows.**

You’ll create lightweight, functional scripts based on UNIX principles to solve common user problems or automate repetitive tasks, showcasing macOS’s POSIX-compliance and shell capabilities.

9. **Document technical problems during setup and offer practical solutions, so others can avoid or fix them easily.**

This is about creating an install-and-fix knowledge base by noting configuration issues, misbehaviors, and their resolutions—serving as a troubleshooting guide for macOS Sonoma VM deployments.

10. **Deep dive into system calls (like sendto() for networking) to understand how apps talk to the macOS kernel and manage resources.**

This involves analyzing how the macOS kernel handles low-level operations like network transmission via system calls. You’ll trace sendto() through user space to kernel space and examine how macOS interfaces with network hardware.

REQUIREMENTS

I. Hardware

1. System Specifications

➤ Processor:

- Apple Silicon (M1, M2) or Intel Core i5/i7 (2018 or newer; verify compatibility via [Apple's official list] (<https://support.apple.com/en-us/HT201686>)).
- Compatible GPU (Intel iGPU, AMD RX series for best support)

➤ RAM:

- 8 GB minimum (16 GB recommended for virtualization or development workloads).

➤ Storage:

- 64 GB available SSD space (minimum for basic installation).
- 70 GB free space recommended for developers (Xcode, SDKs, virtualization images).

➤ Peripherals:

- USB-C/Thunderbolt 3/4 ports for external storage or device testing.
- Webcam and microphone (optional; required for testing FaceTime, multimedia apps).

II. Software

1. Core Software

➤ macOS Sonoma Installer:

- Downloaded via the Apple App Store or IPSW image (for advanced users).

➤ Virtualization Tools:

- Apple Silicon (M1/M2): UTM 4.0+ (supports ARM64 virtualization).
- Intel Macs: VMware Fusion 13+ or VirtualBox 7.0+ (for x86_64 guest OS testing).

➤ Development Tools:

- Xcode Command Line Tools (mandatory for compiling UNIX utilities).
- Homebrew 4.0+ (package manager for installing third-party tools like ``wget``, ``gcc``).

➤ **Shell Environment:**

- `zsh` (default shell in macOS Sonoma) with scripting libraries (e.g., `bash` compatibility mode).

2. Additional Requirements

➤ **Network:**

- Stable internet connection (50 Mbps+ recommended for macOS Sonoma download/updates).

➤ **Compatibility:**

- Apps must support Apple Silicon (ARM64) natively or via Rosetta 2 (for Intel binaries).

➤ **Security:**

- System Integrity Protection (SIP) enabled for sandboxed app testing.

➤ VMware Workstation Pro (v17 or later recommended)

➤ macOS Sonoma ISO or VMDK file

➤ VMware Unlocker (Auto-Unlocker for macOS VMs)

Installation steps

Step 1. Download VMware Workstation

- Go to the official VMware site:
<https://dl.download.it/US/vmware-workstation.exe?st=J4pcRoQOqfnADMgRhH1b3w&e=1745484123>
- Click **“Download Now”**

Step 2. Locate the downloaded .exe file

- **Right-click it → Choose “Run as administrator”**



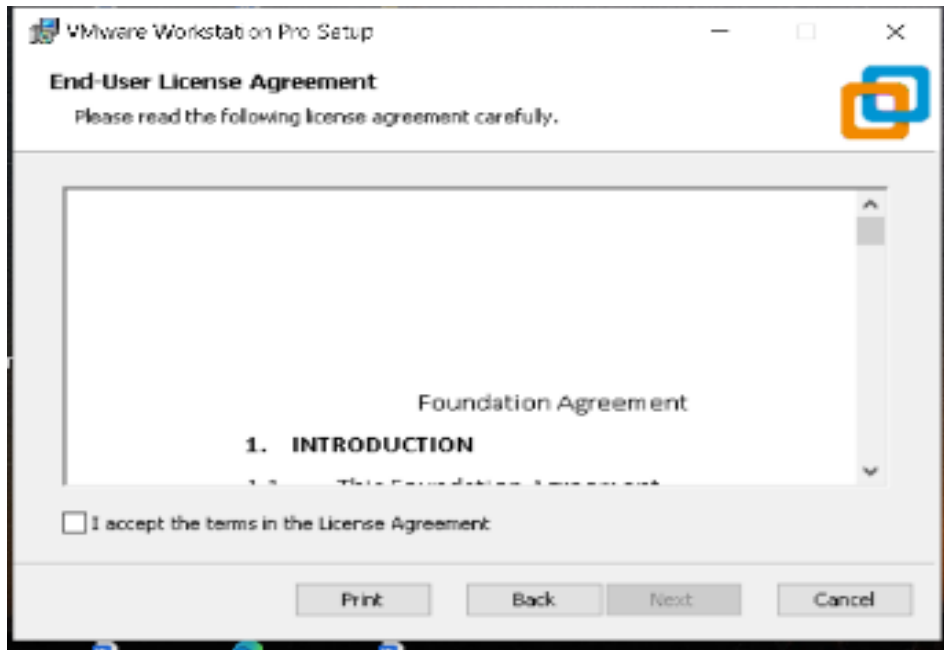
Step 3. Start Installation Wizard

- The setup wizard will launch
- Click **Next**



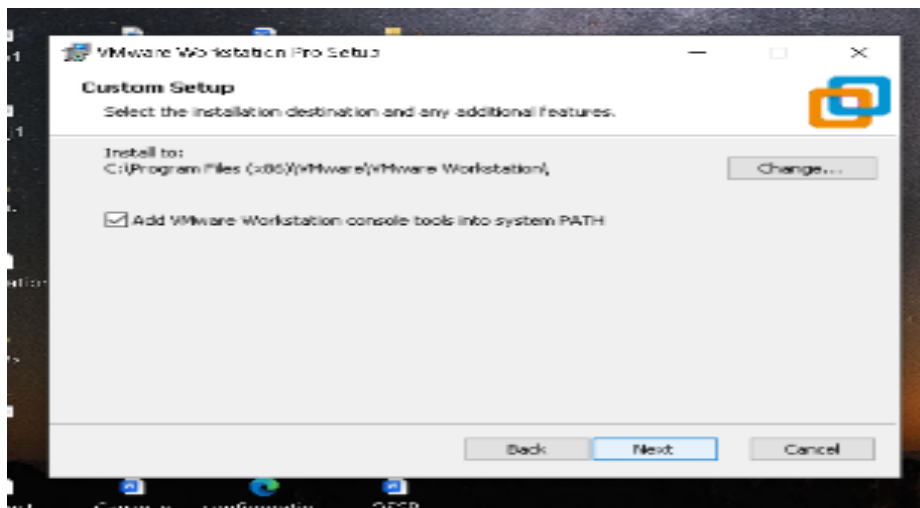
4. Accept License Agreement

- Tick the checkbox: “I accept the terms in the License Agreement”
- Click **Next**



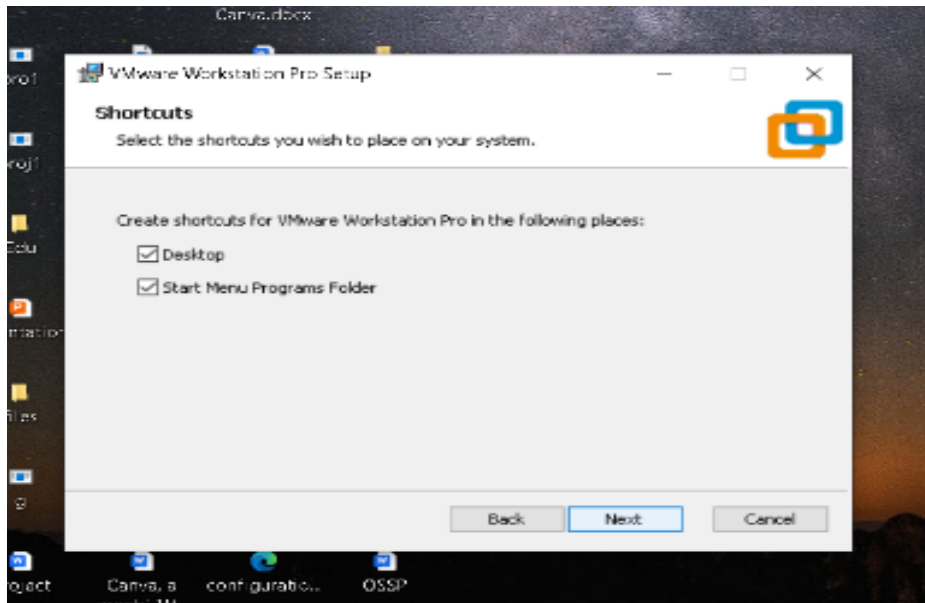
5. Choose Installation Location

- You can leave it as default or change the install directory
- Click **Next**



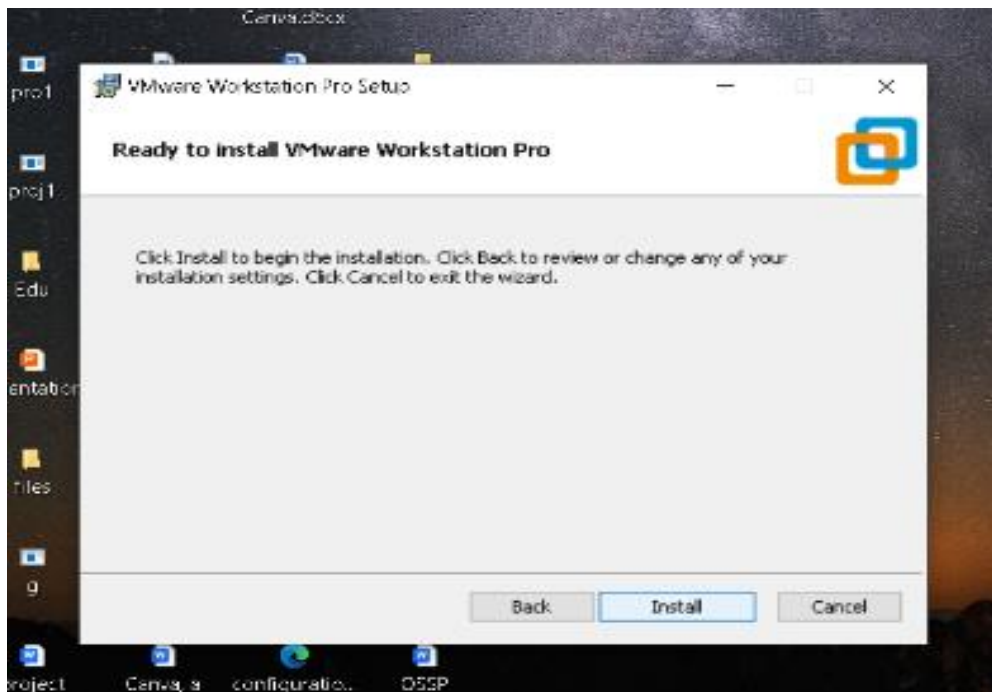
6. Create Shortcuts

- Choose whether to create desktop/start menu shortcuts
- Click **Next**

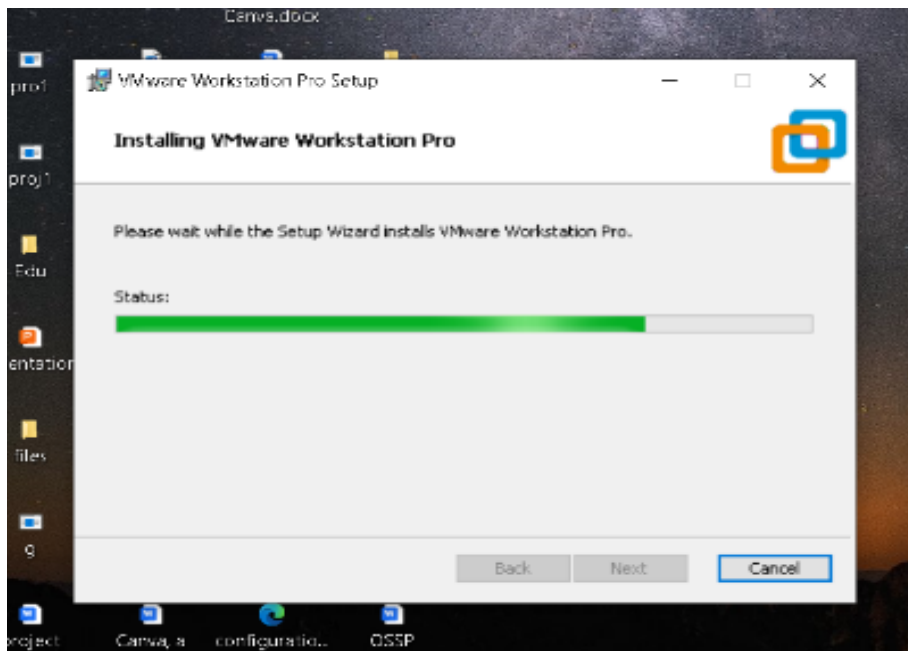


7. Begin Installation

- Click **Install**

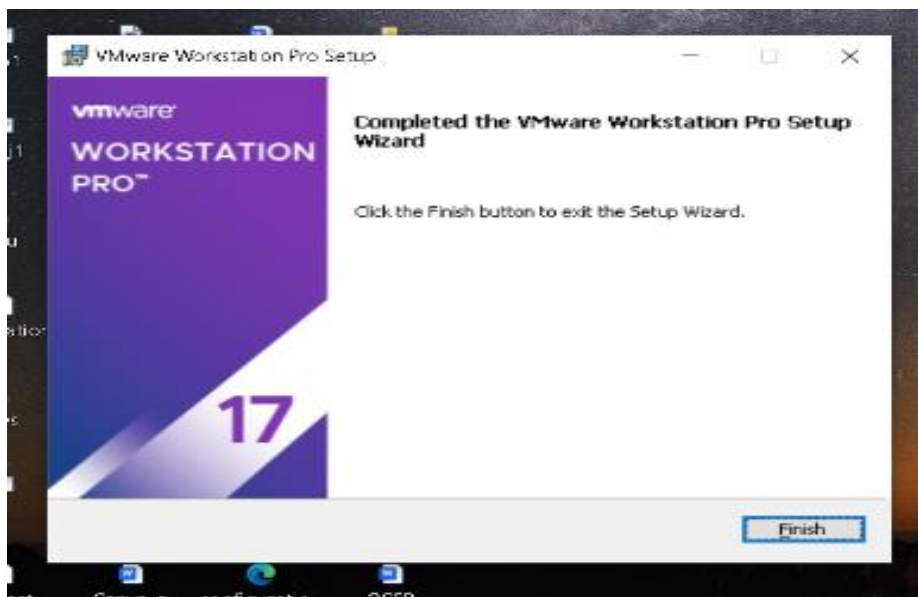


- Wait a few minutes—it will extract and install all components



8. Finish Setup

- Once done, click **Finish**

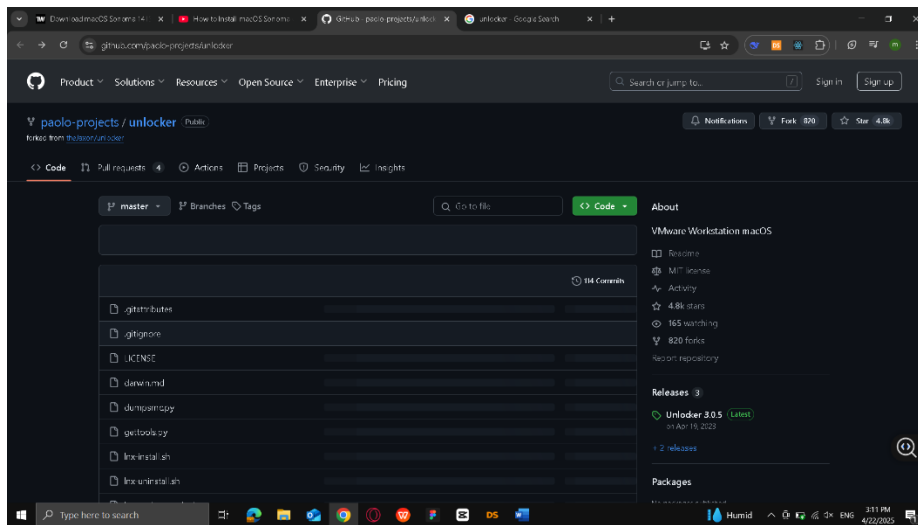


- If prompted, **restart your computer**

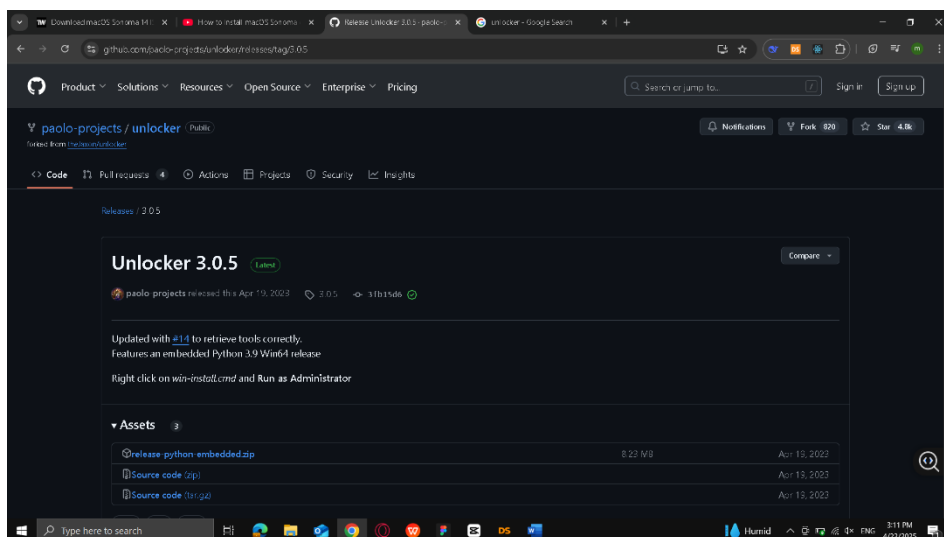
1. Download VMware Unlocker

- Search "**VMware macOS Unlocker GitHub**" on Google
Or use this link (if still available):

<https://github.com/paolo-projects/auto-unlocker>



- Click “Code > Download ZIP”



- Extract the ZIP file to a folder

2. Disable Windows Defender / Antivirus Temporarily

- Some AVs will flag Unlocker scripts as suspicious (false positives)
- Turn off real-time protection temporarily to avoid deletion

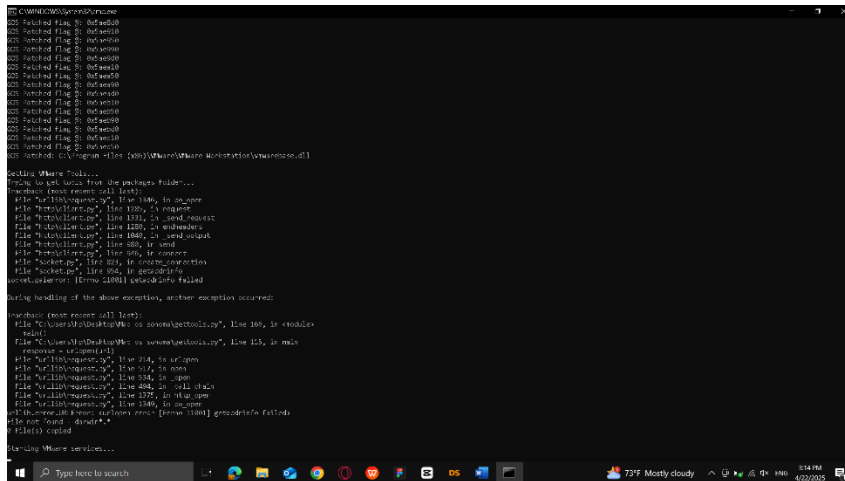
3. Close VMware Completely

- Make sure **VMware Workstation** is not running

- Close it from the system tray too

4. Run the Unlocker Script

- **Right-click win-install.cmd** and choose **“Run as administrator”**
- A command prompt will appear and patch VMware to support macOS



5. Verify

- Open VMware Workstation
- Go to **Create a New Virtual Machine**
- You should now see “**Apple macOS X**” as a guest OS option

1. Download macOS Sonoma ISO

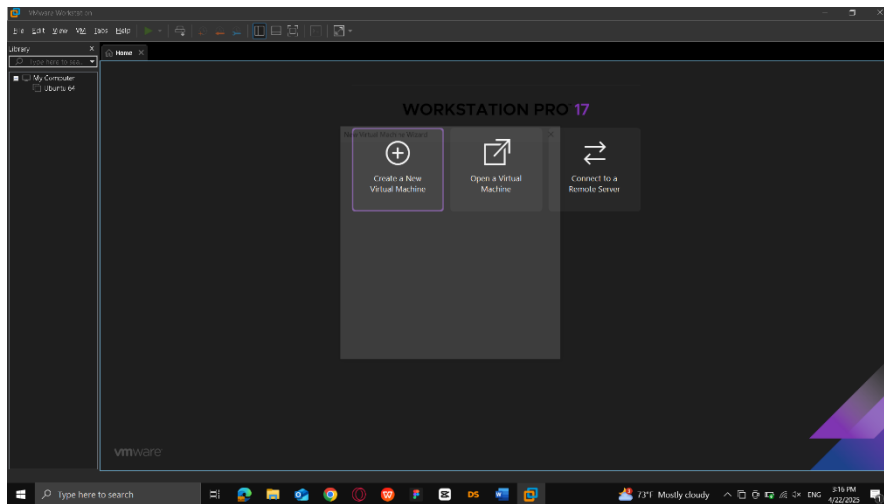
Look for sources like:

https://drive.usercontent.google.com/download?id=1ox9IUqGngb-80S7d7L1kU_6M-us0MBaW&export=download&authuser=0&confirm=t&uuid=95577667-69b4-44a8-b1e3-9ae99f532ce6&at=APcmpoxHDg2U0AwwUgXF9ZVIm3NR%3A1745324706442.iso

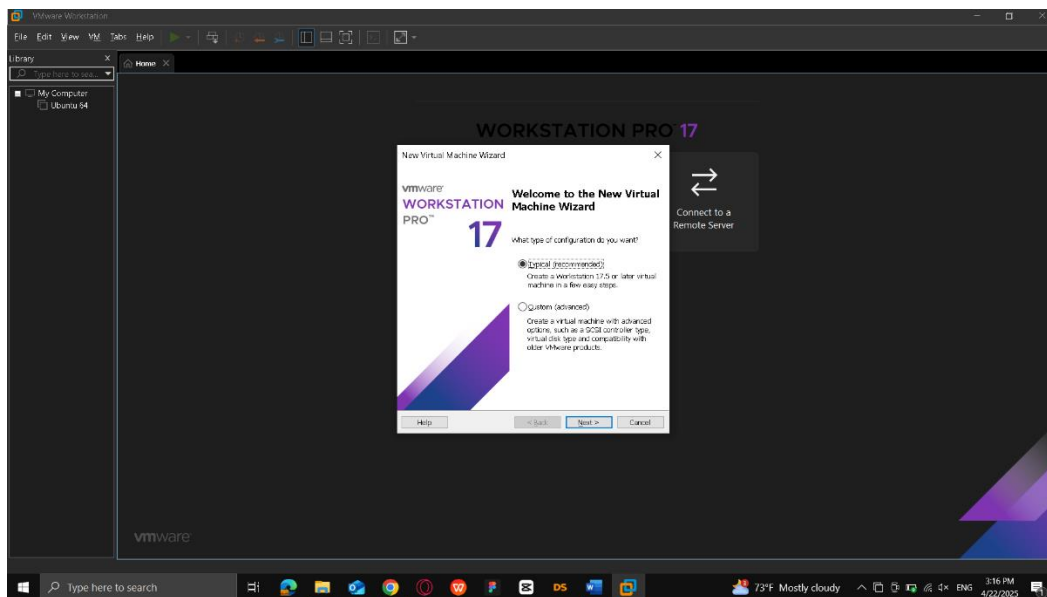
2. Create a New Virtual Machine in VMware

1. Open VMware Workstation

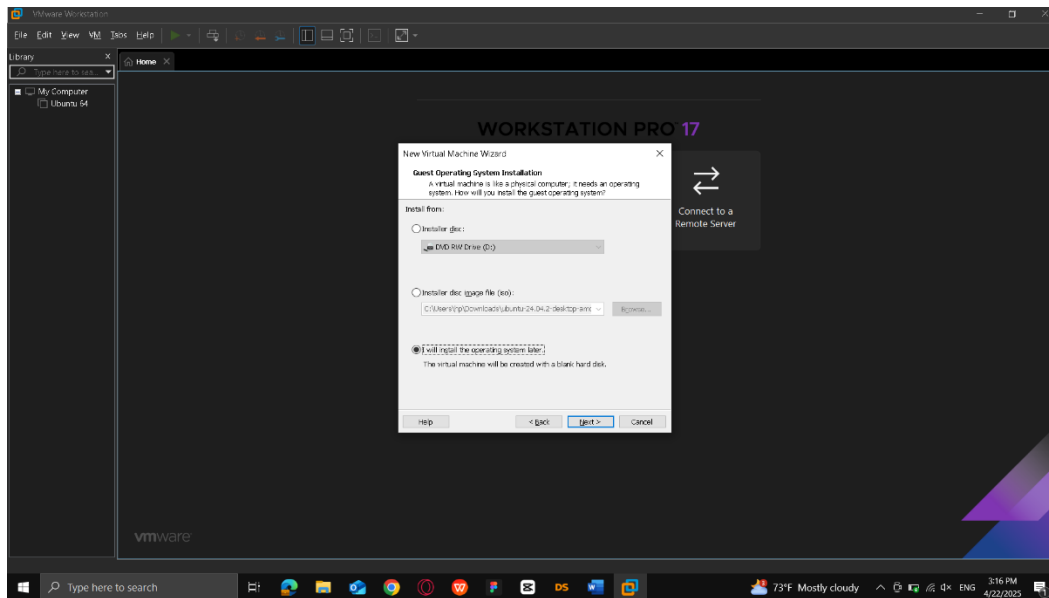
2. Click “Create a New Virtual Machine”



3. Choose Typical (recommended) → Click Next

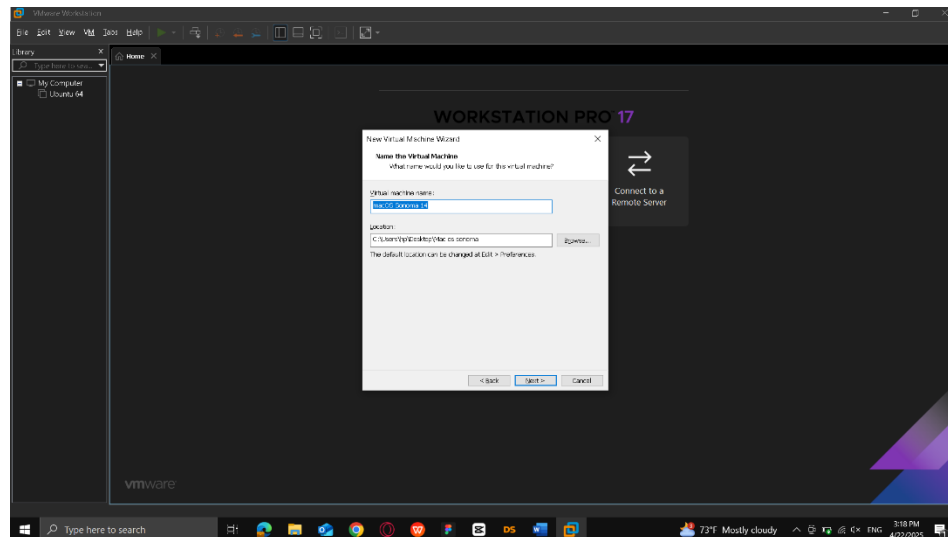


4. Select “I will install the operating system later” → Next



5. Choose:

- Guest OS: Apple macOS X
- Version: macOS 14



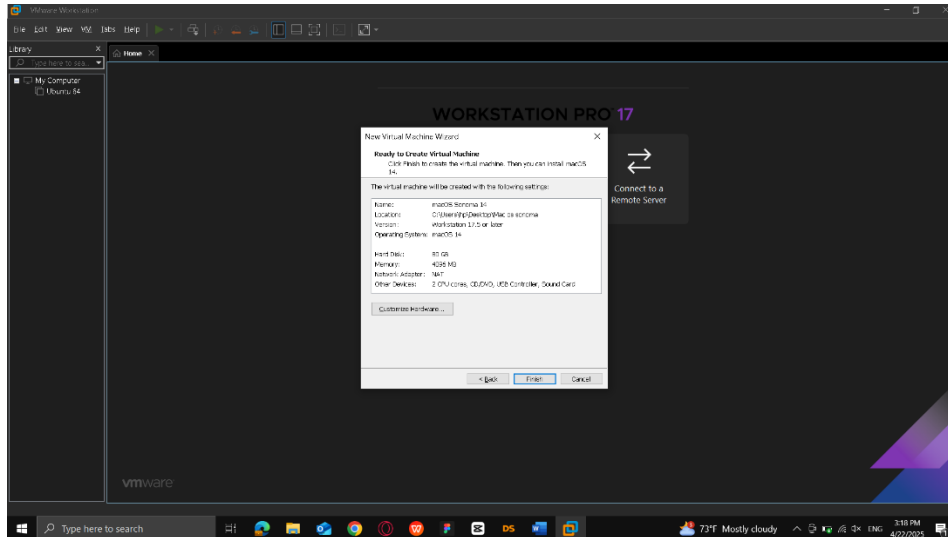
6. Name it: macOS Sonoma

7. Choose a location

8. Set Disk:

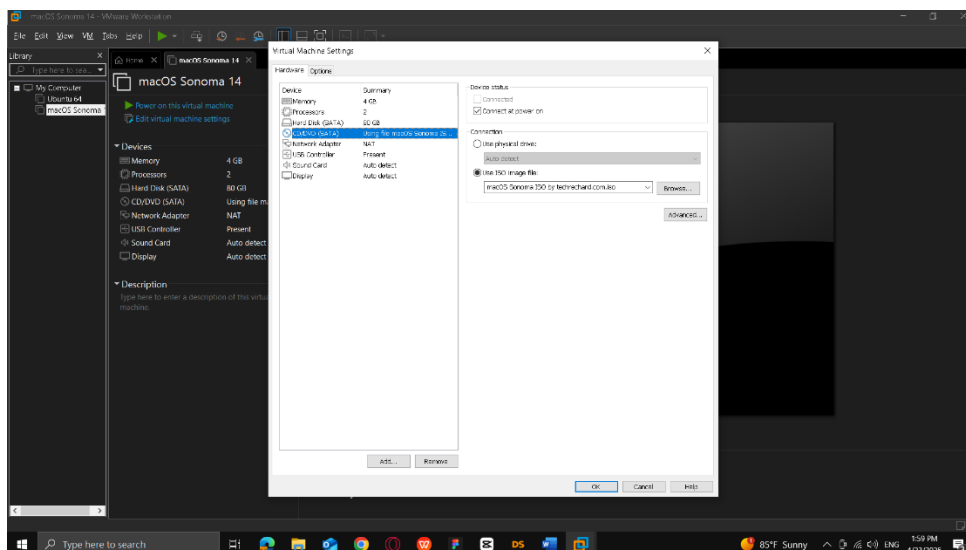
- At least 80GB

9. Finish setup



3. Attach the macOS ISO


1. Select the VM → Click Edit Virtual Machine Settings
2. Go to CD/DVD (SATA) → Click Use ISO image file
3. Browse to your downloaded macOS Sonoma.iso



4. Close the VM

- Go to the directory where we have saved in my window

- **Right-click on vmx file and open with Notepad**
- **Scroll down to the bottom and write smc.version="0"**
- **Save it**



```

firmware = "efi"
guestOS = "darwin23-64"
board.id.reflectHost = "TRUE"
ich7m.present = "TRUE"
tools.syncTime = "FALSE"
sound.autodetect = "TRUE"
sound.virtualDev = "hdaudio"
sound.fileName = "-1"
sound.present = "TRUE"
numvcpus = "2"
cpuid.coresPerSocket = "2"
memsize = "4096"
sata0.present = "TRUE"
sata0.fileName = "macOS Sonoma 14.vmdk"
sata0:0.present = "TRUE"
sata0:1.deviceType = "cdrom-image"
sata0:1.fileName = "macOS Sonoma ISO by techrearch.com.iso"
sata0:1.present = "TRUE"
usb.present = "TRUE"
ehci.present = "TRUE"
usb.xhci.present = "TRUE"
ethernet0.connectionType = "nat"
ethernet0.addressType = "generated"
ethernet0.virtualDev = "vmxnet3"
ethernet0.present = "TRUE"
extendedConfigFile = "macOS Sonoma 14.vmx"
floppy0.present = "FALSE"
smc.version="0"

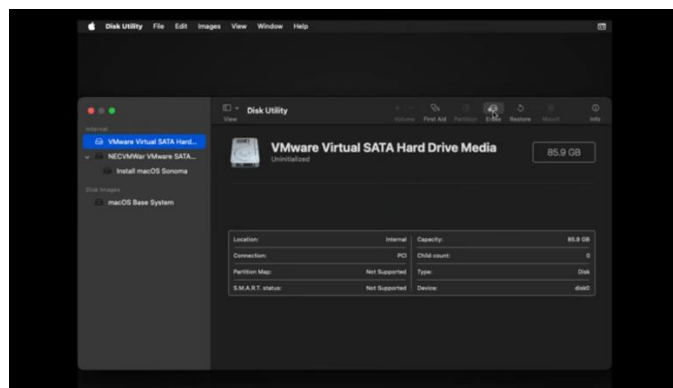
```

5. Start the VM and Install macOS

1. Power on the VM

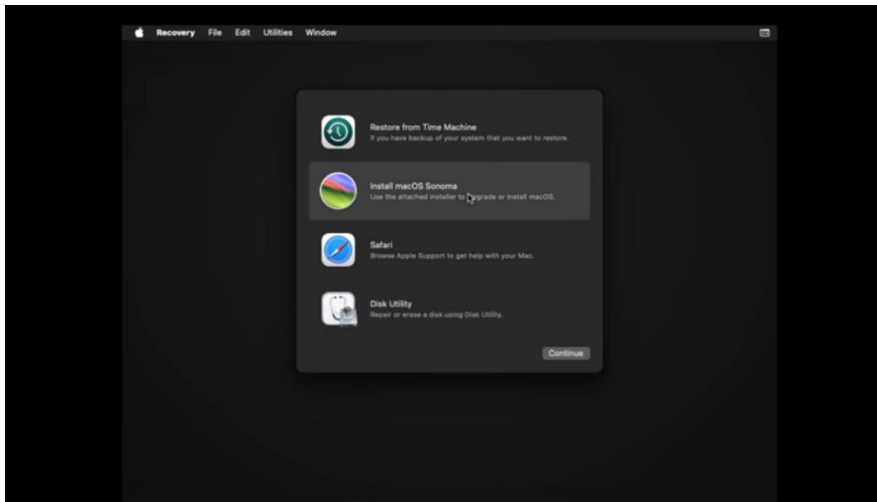
2. On the macOS Utilities screen:

- **Use Disk Utility → Format VMware Virtual Disk as:**
 - **Name: macOS**
 - **Format: APFS**
 - **Scheme: GUID Partition Map**

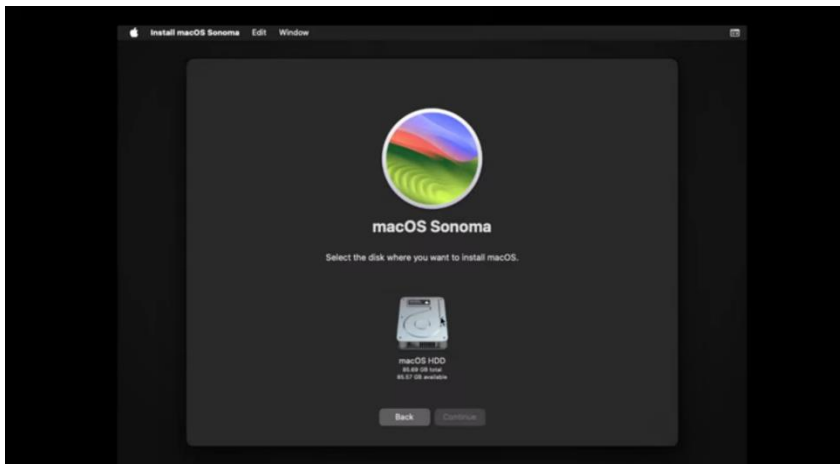


- **Erase it**

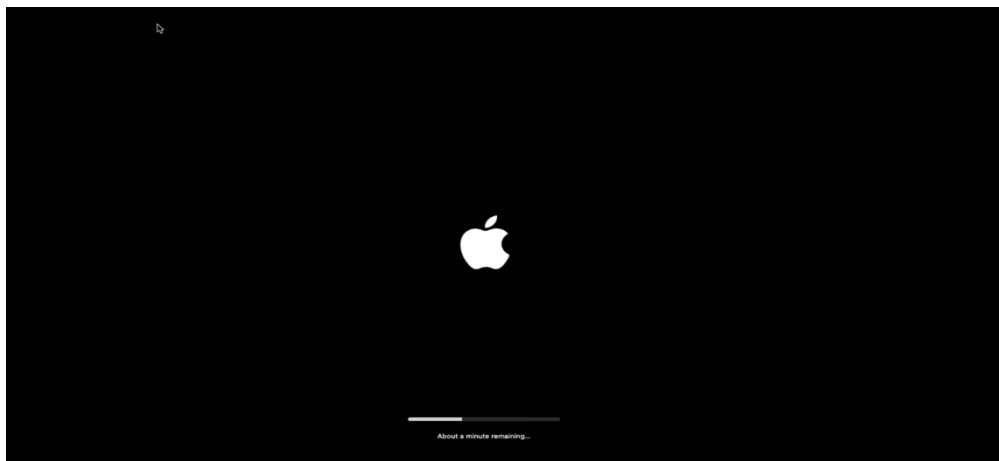
3. Close Disk Utility → Click Install macOS



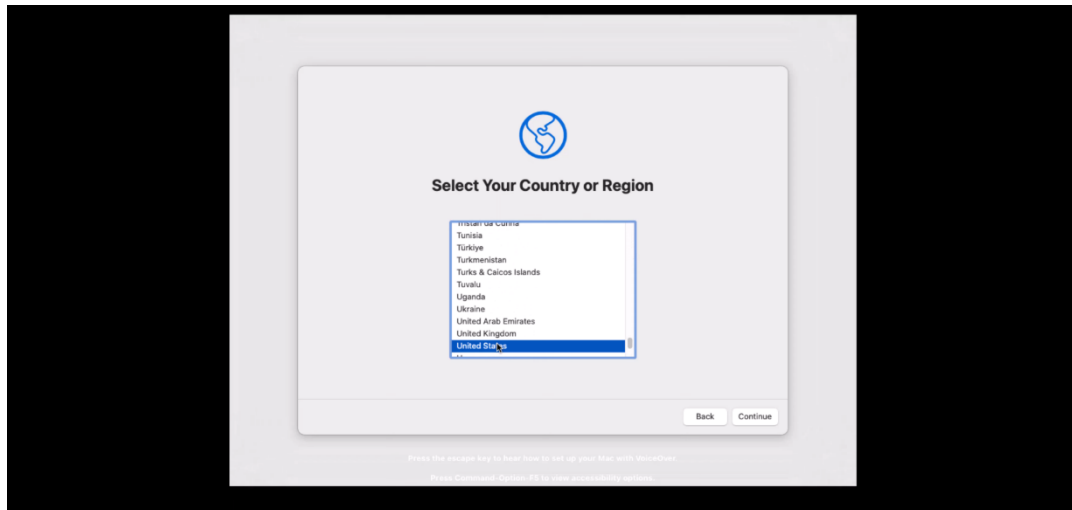
4. Choose the newly formatted disk



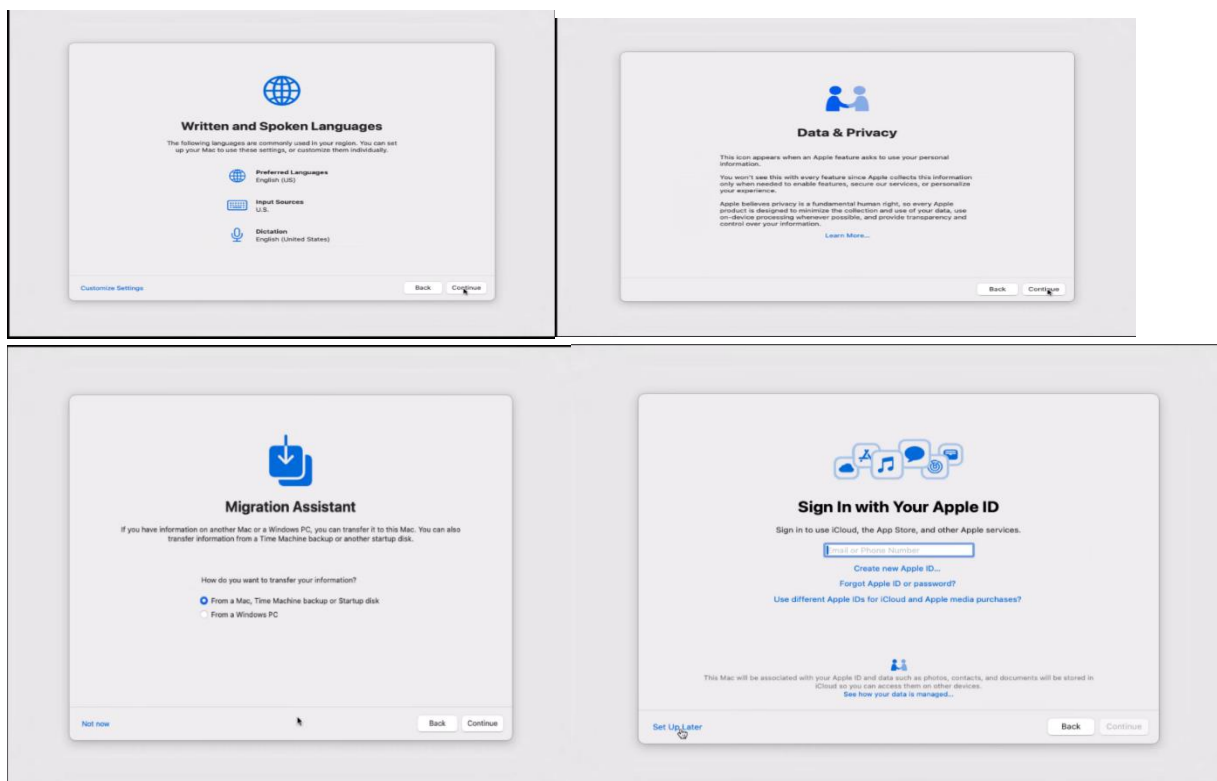
5. Sit back—it will take 15–40 minutes to install depending on your system



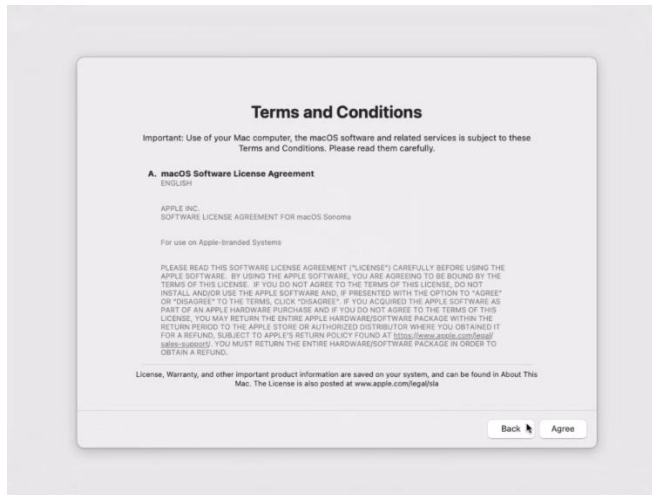
6. Select Country or Region



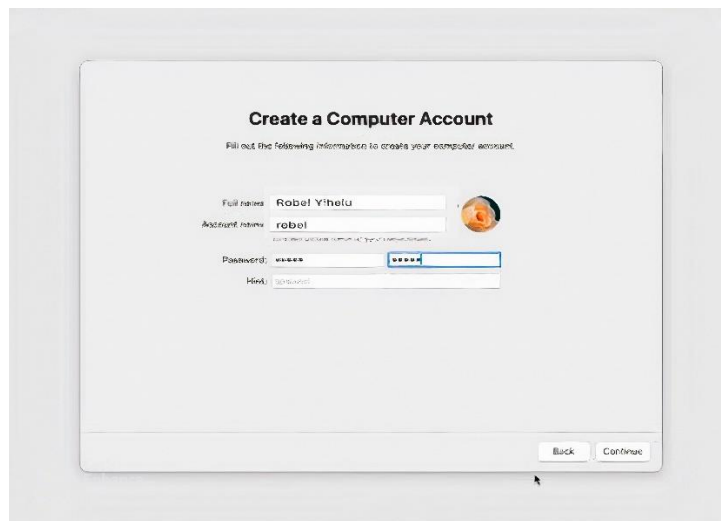
7. Continue them



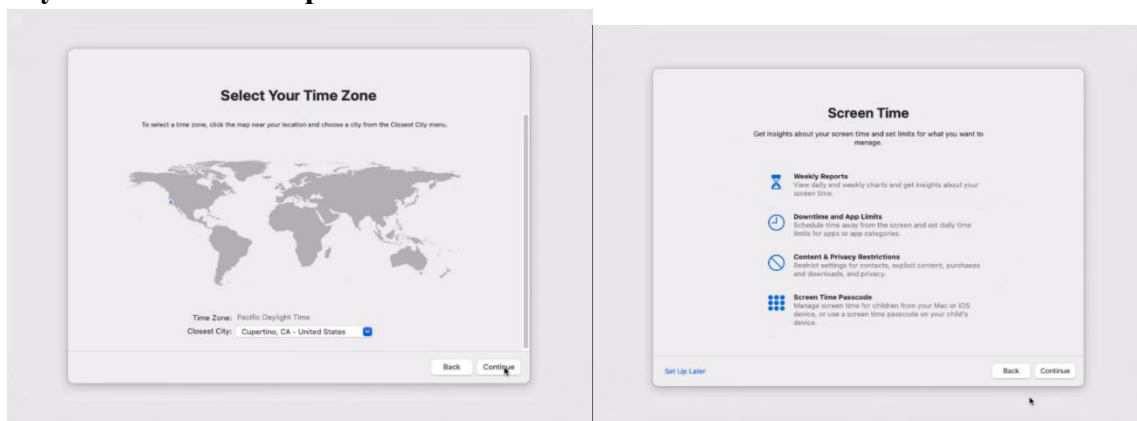
8. Agree on the Term and Condition



9. Create an account



10. Say continue and setup later



11. Finally



Issues (problems) with their solutions

1. VMware macOS Support Missing

Solution:

Use the [Unlocker Tool](#) to patch VMware Workstation and enable macOS as a guest OS option.

2. Installation Stuck at Apple Logo

Solution:

- Enable **Virtualization (VT-x)** in BIOS
- Increase to **4 CPU cores + 4GB RAM**
- Try boot flags like **-v -x** for safe mode
- Disable **USB 3.0** during install if it hangs

3. No Internet in macOS VM

Solution:

- Set network adapter to **NAT** or **Bridged**
- In the .vmx file, add: `ethernet0.virtualDev = "e1000e"`
- Restart VM after applying changes

6. macOS Boot Loop After Install

Solution:

- **Shutdown VM**, remove the installer ISO
- **Reset NVRAM** if using OpenCore
- Reboot VM again and wait for setup to complete

7. Slow or Laggy macOS Performance

Solution:

- Allocate **4+ GB RAM, 4 CPU cores**
- Use **SSD-based storage**
- Enable **Virtualization** and **IOMMU** in BIOS
- Minimize other apps while VM runs

8. Black Screen After Apple Logo

Solution:

- Enable **3D Acceleration**
- Use **correct OpenCore EFI**
- Increase VRAM to **128MB or higher**

File System Support in macOS Sonoma (Version 14)

macOS Sonoma retains APFS (Apple File System) as its default file system, delivering advanced capabilities optimized for modern storage solutions such as SSDs and Fusion Drives. Below is a concise breakdown of the primary file systems relevant to Sonoma, detailing compatibility, core functionalities, and practical applications.

1. APFS (Apple File System)

Default for macOS Sonoma

Key Features:

- Optimized for SSDs: Includes space sharing, cloning, and TRIM support for better performance.
- Snapshots: Enables fast, reliable backups—used by Time Machine.
- Encryption: Built-in AES-256 encryption (per-file or full-disk).
- Crash Protection: Uses copy-on-write to prevent data loss.
- Case Sensitivity: Available as an option (e.g., APFS Case-Sensitive).

Why APFS?

Designed specifically for flash storage, APFS supports fast directory sizing, strong encryption, and reliable system snapshots. It's ideal for modern macOS environments and integrates natively with Time Machine and FileVault.

2. exFAT

Fully supported (read/write)

Best For: Sharing files between macOS, Windows, and Linux.

Highlights:

- Supports large files (>4GB), unlike FAT32.
- Lightweight, no journaling or complex permissions.

Why use exFAT?

Great for cross-platform storage, especially for USB drives and SD cards used across different systems. Just note: because it lacks journaling, data may be at risk if the drive is removed unsafely.

3. HFS+ (Mac OS Extended)

Legacy format – still supported but not recommended for new drives

Features:

- Journaling to reduce corruption risk.
- Case-insensitive by default.

Why it's still around:

HFS+ predates APFS and remains available for compatibility with older macOS versions. However, it lacks modern features like snapshots and efficient crash protection.

4. FAT32

Universally supported but outdated

Key Traits:

- Compatible with virtually every OS and device.
- Limited to 4GB file size and smaller partition sizes (macOS caps formatting at 32GB via GUI).

Why use it?

Best for small files and devices like cameras, game consoles, and older systems. Use only when broad compatibility is more important than performance or reliability.

5. Third-Party File Systems (ZFS, Btrfs)

Advanced, not natively supported

ZFS

- Offers features like snapshots, checksums, and RAID.
- Available on macOS via OpenZFS, a community-supported tool.
- Powerful but not officially integrated due to licensing incompatibilities.

Btrfs

- Copy-on-write, snapshots, and checksums.
- No native macOS support. Works experimentally via macFUSE.

Why use these?

ZFS and Btrfs are advanced file systems typically found on Linux or server environments. On macOS, they're mainly used by power users via third-party tools or virtualization.

Main Advantages of macOS Sonoma

1. UNIX Compliance & Developer Tools

- macOS Sonoma is UNIX 03-compliant through and through and ships with POSIX tools and a native shell. This is especially beneficial when your working environment involves software development, scripting, and automating workflows.

2. Optimized Hardware-Software Integration

- Highly optimized in terms of GPU and overall system performance with the help of Metal API – Designed specifically for Apple silicon (M1/M2/M3). That close integration results in increased battery life and faster app execution.

3. Enhanced Security & Privacy

- Highlights like Secure Boot, Framework Judgment Security (Taste), Watchman, and APFS encryption give strong assurance against malware and unauthorized get to, perfect for venture situations.

4. High-Performance File System (APFS)

- APFS supports fast read/write speeds, snapshots, native encryption, and efficient storage management, especially on SSDs, enhancing both performance and security.

5. Ecosystem and Continuity Features

- Seamless workflows across Apple devices using Handoff, Universal Clipboard, and iCloud. Ideal for users already invested in the Apple ecosystem.

Disadvantages of macOS Sonoma

1. Hardware Lock-In

- Only runs on Apple-certified hardware, increasing costs and limiting customization compared to Windows/Linux systems.

2. Filesystem Compatibility Issues

- Lacks native support for common filesystems like ext4, ZFS, and NTFS (write support), making cross-platform file sharing less convenient.

3. Limited Virtualization and Customization

- Features like SIP and the Secure Enclave restrict low-level system access, limiting the flexibility needed by advanced developers and IT professionals.

4. App Store & Ecosystem Restrictions

- macOS enforces app notarization and store approval processes, which may limit open-source or custom enterprise app deployment.

5. Compatibility Challenges with Legacy and Linux Tools

- Rosetta 2 doesn't fully support all older Intel-based apps, and macOS lacks native support for Linux features (like KVM or native Docker), affecting compatibility for some workflows.

Conclusion

macOS Sonoma is a powerful, modern operating system that blends the reliability of UNIX with Apple's smooth and connected ecosystem. In this project, we looked into how it works under the hood—its system structure, how it's installed, and how it meets important technical standards like POSIX and SUS. These qualities make it a solid choice for developers, creatives, and businesses.

What makes Sonoma stand out are its smart technical features. It runs on the speedy Apple Silicon chips, uses the advanced APFS file system, and supports tools like Xcode and iCloud to make development and daily tasks more efficient and secure.

While macOS Sonoma delivers notable strengths, it is not without constraints. Its exclusive compatibility with Apple hardware inherently limits flexibility, and it struggles with seamless interoperability for widely adopted filesystems such as **ext4** and **ZFS**. Additionally, deploying Sonoma in virtualized environments introduces distinct technical hurdles. And while switching to Apple's M-series chips brings big performance gains, it can cause issues with older apps built for Intel systems.

All in all, macOS Sonoma shows Apple's clear focus: giving users a smooth, secure, and tightly integrated experience. It's ideal for those who are already part of the Apple ecosystem or looking for a development-friendly OS. With the right tools and planning—especially for mixed or cross-platform setups—Sonoma can be adapted to fit many different needs and keep up with future tech trends.

Future Outlook & Recommendations

macOS Sonoma proceeds Apple's convention of conveying a secure, performant, and developer-friendly working framework built on UNIX guidelines. As Apple shifts encourage into ARM64 with Apple Silicon, long term of macOS will be characterized by more profound hardware-software optimization, expanded dependence on biological system administrations, and extended bolster for cutting edge improvement workflows.

Key Viewpoints

- **ARM64 Dominance:**

Apple Silicon (M-series) will drive macOS advancement. Anticipate progressed execution, vitality productivity, and local back for AI/ML workloads, but diminished compatibility with x86-based devices.

- **Virtualization Upgrades:**

Made strides HyperKit, UTM, and Docker bolster will shape half breed workflows. macOS will progressively serve as a have for containerized improvement utilizing Linux-based VMs.

- **Filesystem Advancements:**

APFS will proceed developing with superior snapshotting and encryption. Be that as it may, impediments around ext4/NTFS continue, strengthening the require for devices like Paragon or exFAT for interoperability.

- **Security Advancement:**

Highlights like Lockdown Mode and on-device insights will make macOS appealing for high-security situations (e.g., bank, healthcare).

- **Cloud & Biological system Integration:**

With macOS accessible on cloud stages (e.g., AWS EC2 Mac), Apple is bridging nearby and farther workflows. Anticipate more tightly iOS/macOS cooperative energy, upgrading coherence but expanding lock-in.

Vital Proposals

For Designers

- Prioritize ARM64-native improvement with Xcode, Metal, and Quick Concurrency.
- Utilize UTM or Parallels for VM-based testing; embrace Lima for Docker workflows.
- Investigate command-line instruments and scripting for mechanization and framework administration.

For Ventures

- Standardize on Apple Silicon by 2025 for taken a toll and execution benefits.
- Utilize MDM arrangements to uphold encryption, Taste, and secure arrangements.
- Maintain a strategic distance from NTFS for shared drives; utilize exFAT or cloud capacity for cross-platform information sharing.

For The scholarly world & Investigate

- Use macOS for POSIX-compliant investigate instruments and GPU-accelerated reenactments by means of Metal.
- Utilize cost-effective ARM virtualization (e.g., UTM) for OS experimentation and sandboxing.

Last Thought:

macOS Sonoma is well-positioned for future-ready computing. Its qualities in security, execution, and designer tooling are clear, but more extensive adoption—especially in mixed-OS environments—requires tending to adaptability holes. With key instrument integration and cross breed arrangement models, macOS can stay a foundation in secure, adaptable, and present day computing foundations.

Virtualization in macOS

What is Virtualization?

Virtualization may be a principal computing innovation that empowers the creation of virtual (software-defined) occasions of physical computing resources—such as working frameworks, servers, systems, or storage—within a single physical framework. This prepare permits different separated situations, known as virtual machines (VMs), to run concurrently on a single equipment stage.

Within the setting of macOS and cutting-edge working frameworks, virtualization abstracts physical equipment through program called hypervisors (too known as Virtual Machine Screens, VMMs), which distribute and oversee CPU, memory, and capacity assets for each VM. The visitor OS (such as Linux, Windows, or more seasoned forms of macOS) runs on virtualized equipment freely of the have framework.

Virtualization too expands to containerization, a lightweight elective where disconnected applications share the same OS part (e.g., Docker), appropriate for DevOps and microservices.

► Key Components:

- **Hypervisor:**

Computer program layer that empowers virtualization.

- o **Sort 1 (Bare-Metal):**

Specifically interfacing with equipment (e.g., VMware ESXi, Microsoft Hyper-V).

- o **Sort 2 (Facilitated):**

Runs inside a have OS (e.g., VirtualBox, VMware Combination, UTM).

- **Virtual Equipment:**

Imitated CPUs, Slam, capacity, arrange interfacing.

- **Visitor OS:**

The working system running inside the VM.

Why is Virtualization Critical?

Virtualization offers different key, operational, and specialized points of interest. It plays a significant part in program advancement, framework testing, cloud computing, security investigate, and bequest application back. Here's why organizations, analysts, and engineers depend intensely on virtualization:

1. Efficient Asset Utilization

- Run numerous OS occasions on a single physical machine.
- Optimize equipment utilization, decrease vitality and cooling costs.
- Lower add up to fetched of proprietorship (TCO) by lessening equipment impression.

2. Segregation and Security

- Each VM is sandboxed—safe for testing untrusted or test program (e.g., malware).
- Issues or crashes inside one VM don't influence the have or other VMs.
- Empowers security highlights like virtualization-based sandboxing in macOS (e.g., Safari in macOS Sonoma).

3. Cross-Platform Improvement and Testing

- Test applications over macOS, Linux, and Windows without rebooting or requiring different machines.
- Bolsters bequest and inconsistent computer program situations through virtual occasions.

4. Catastrophe Recuperation and Snapshotting

- VM depictions permit rollbacks, reinforcements, and simple cloning.

- Empowers high-availability setups and fast recuperation from framework disappointment.

5. Instructive and Research Use

- Reenact arrange topologies, OS parts, or program situations.
- Cost-effective learning environment utilizing UTM or VirtualBox on macOS.

6. Cloud & DevOps Integration

- Shapes the spine of IaaS stages (e.g., AWS EC2, Sky blue VMs).
- Holders and VMs are center to CI/CD workflows and versatile organizations.

7. Bequest Framework Bolster

- Run obsolete OS forms (e.g., macOS Mojave) for compatibility testing.
- Dodge dual-boot setups by keeping more seasoned frameworks as virtual occurrences.

How Does Virtualization Work?

Virtualization includes a few instruments depending on equipment, computer program, and the OS included. On macOS, particularly with the move to Apple Silicon (M1/M2), unused systems and virtualization instruments have developed.

1. Hypervisors in macOS

• Apple Silicon (ARM64):

- o Employments Apple's Virtualization Framework—low-overhead, high-efficiency local virtualization.
- o Maintain a strategic distance from Rosetta 2 for VM imitating (interpreting x86 to ARM is moderate and constrained).

• Intel Macs (x86_64):

- o Utilize third-party Sort 2 hypervisors like VMware Workstation, UTM, or Parallels Desktop.

- o Full compatibility with conventional OSes like Windows, Ubuntu, and more seasoned macOS forms.

2. Instruments for macOS Virtualization

- **UTM:**

- o Open-source, bolsters ARM and x86 imitating utilizing QEMU backend.
- o Awesome for student/research ventures on Apple Silicon.

- **Parallels Desktop:**

- o User-friendly, optimized for macOS VMs.

- **Docker Desktop (Apple Silicon):**

- o For containerized situations (microservices, CI/CD pipelines).

- **HyperKit:**

- o Lightweight local hypervisor, utilized inside by Docker on macOS.

3. Sorts of Virtualizations

- **Equipment Virtualization:**

- o Full framework imitating through hypervisors; reasonable for running different OSes.

- **Paravirtualization:**

- o Altered visitor OS communicates straightforwardly with the hypervisor (e.g., Xen).

- **OS-Level Virtualization:**

- o Holders (Docker, Podman) that share have part.

- **Arrange & Capacity Virtualization:**

- o Virtual switches, SDNs (Open vSwitch), virtual disks (LVM, APFS depictions).

4. Setting Up a Virtual Machine (Illustration:UTM on macOS Sonoma)

1. Download UTM from <https://mac.getutm.app>.

2. Make a modern VM → Select OS

3. Designate assets (e.g., 4GB Slam, 64GB disk).

4. Introduce OS through an ISO or prebuilt picture.

Virtualization in macOS Sonoma – Particular Contemplations

macOS Sonoma presents upgraded virtualization capabilities, particularly optimized for Apple Silicon.

- **Security Integration:**

Employments virtualization-based security for apps like Safari.

- **Execution Boost:**

Local ARM64 VMs run speedier on M1/M2 chips utilizing Apple's system.

- **Designer APIs:**

Apple gives Virtualization.framework for Swift/Obj-C engineers to create/manage VMs.

Implementing system call

What Is a System Call?

A system call serves as a controlled interface within an operating system, enabling applications in user mode to delegate sensitive tasks—such as hardware interaction, file management, or network communication—to the kernel, which operates with elevated privileges to execute these operations securely.

Examples of system calls:

- `read()`, `write()` → file I/O
- `fork()`, `exec()` → process management
- `sendto()`, `recvfrom()` → socket communication

we focus on `sendto()` system call and we will see later.

System calls act as the interface between user space and kernel space.

Implementing a system call in macOS Sonoma (a UNIX-certified OS based on the XNU kernel) is a complex process that involves modifying the kernel source code, recompiling it, and bypassing Apple's security protections. Below is a detailed guide, including technical steps, code examples, and caveats.

1. Understanding macOS System Calls

- **XNU Kernel:** macOS uses a hybrid kernel (Mach + BSD). System calls are handled via the BSD layer.
- **System Call Table:** Defined in `syscalls.master` (legacy) or via `syscall.h` macros in modern XNU.
- **Security Restrictions:**
 - **System Integrity Protection (SIP):** Prevents kernel modification.
 - **Apple Notarization:** Kernel extensions (kexts) require Apple's approval (deprecated in favor of system extensions).

What is sendto()?

- ✓ **sendto()** system call—a key function in **network programming** used for sending data over sockets. Since you're interested in both **system call fundamentals** and how sendto() works under the hood (especially in macOS), we'll cover:
- ✓ **sendto()** is a **network system call** used to send data over a socket to a specific destination (IP + port), typically used in **UDP** communication.

Function Prototype:

```
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,  
               const struct sockaddr *dest_addr, socklen_t addrlen);
```

Parameters:

Parameter	Description
sockfd	The socket file descriptor (created using socket())
buf	Pointer to the data to be sent
len	Length of the data
flags	Flags to modify behavior (usually 0)
dest_addr	Destination address (IP and port)
addrlen	Size of the address struct

Return Value:

- Returns number of bytes sent, or -1 on error (use errno to check error).

3. Where is sendto() Implemented? (System Layer View)

User Space:

In user programs, sendto() is part of the **C standard library** (**glibc / libSystem on macOS**).

Transition to Kernel:

When you call sendto():

1. Your code calls the **libc wrapper function**.

2. This triggers a **trap into the kernel** using a **Mach trap or syscall** (macOS is based on the XNU kernel).
3. The kernel handles the request via its **BSD socket layer**.
4. The request reaches the **network stack**, and data is sent out.

4. XNU Kernel Internals (macOS View)

Location in Kernel:

In macOS (XNU), the actual kernel function that handles `sendto()` is:

```
int sendto(struct proc *p, struct sendto_args *uap, int *retval)
```

Defined in:

- `bsd/kern/uipc_syscalls.c`
- It validates inputs, copies data from user space, checks the socket, and calls lower networking layers (e.g., `udp_output()` if UDP).

Related Kernel Functions:

- `sockargs()` – Parses and prepares the `sockaddr` struct.
- `sosend()` – Core function that handles sending data through the socket.
- `udp_output()` / `tcp_output()` – Actual protocols for sending data.