Sample Code

Running macOS in a virtual machine on Apple silicon

Install and run macOS in a virtual machine using the Virtualization framework.

Download

macOS 14.0+ | Xcode 26.0+

Overview

This sample code project demonstrates how to install and run macOS virtual machines (VMs) on Apple silicon. The Xcode project includes two separate apps:

- `InstallationTool`, a command line utility that installs macOS from a restore image, which is a file with a `.ipsw` file extension, onto a VM. You can use this tool to download the restore image of the most current macOS release from the network, or with your own restore image. The utility creates a VM bundle and stores the resulting VM images in your Home directory.
- `macOSVirtualMachineSampleApp` is a Mac app that runs the macOS VM that `InstallationTool` installs. You use `macOSVirtualMachineSampleApp` to launch and control the macOS VM that loads and runs macOS from the VM bundle. This app includes entitlements to enable it to use Virtualization and access the VM's' audio input, such as the microphone.

There are four build targets in this project that represent the `InstallationTool` and the `macOSVirtualMachineSampleApp`, one set of targets each for Swift and Objective-C versions of the apps. You can use either version, they're functionally identical.

> **Note**
>
> The default deployment target is macOS 14. If you need to build for an earlier version of macOS, you need to change the deployment target as appropriate.

Configure the sample code project

You need to install the virtual machine, and `VM.bundle` needs exist before launching the sample app.

1. Set up code signing for each of the project's targets by navigating to the Signing & Capabilities settings and selecting your team from the drop-down menu.
2. Run `InstallationTool` from within Xcode or in Terminal to download the latest available macOS restore image from the network and create a macOS VM image on disk.

   `InstallationTool` creates a `VM.bundle` package in your Home directory, containing:

   - `Disk.img` — The main disk image of the installed OS.
   - `AuxiliaryStorage` — The auxiliary storage for macOS.
   - `MachineIdentifier` — The data representation of the `VZMacMachineIdentifier` object.
   - `HardwareModel` — The data representation of the `VZMacHardwareModel` object.
   - `RestoreImage.ipsw` — The restore image downloaded from the network (this file exists only if the tool runs without arguments).
3. Launch `macOSVirtualMachineSampleApp` to run the macOS guest operating system. The sample app starts the VM and configures a graphical view that you interact with. The virtual Mac continues running until you shut it down from inside the guest OS, or quit the app.

   To reinstall the virtual machine, delete the `VM.bundle` package and run `InstallationTool` again.

Install macOS from a restore image

After downloading a restore image, you can install macOS from that restore image.

Swift | Objective-C

```
let installer = VZMacOSInstaller(virtualMachine: virtualMachine, restoringFromImageA
```

```
NSLog("Starting installation.")
installer.install(completionHandler: { (result: Result<Void, Error>) in
    if case let .failure(error) = result {
        fatalError(error.localizedDescription)
    } else {
        NSLog("Installation succeeded.")
    }
})

// Observe installation progress.
installationObserver = installer.progress.observe(\.fractionCompleted, options: [.ir
    NSLog("Installation progress: \(change.newValue! * 100).")
}
```

Set up the virtual machine

The sample app uses a VZVirtualMachineConfiguration object to configure the basic characteristics of the guest, such as the CPU count, memory size, various device configurations, and a VZMacOSBootLoader object to load the operating system from the disk image, as the following example shows:

Swift | Objective-C

```
let virtualMachineConfiguration = VZVirtualMachineConfiguration()

virtualMachineConfiguration.platform = createMacPlaform()
virtualMachineConfiguration.bootLoader = MacOSVirtualMachineConfigurationHelper.crea
virtualMachineConfiguration.cpuCount = MacOSVirtualMachineConfigurationHelper.comput
virtualMachineConfiguration.memorySize = MacOSVirtualMachineConfigurationHelper.comp

virtualMachineConfiguration.audioDevices = [MacOSVirtualMachineConfigurationHelper.c
virtualMachineConfiguration.graphicsDevices = [MacOSVirtualMachineConfigurationHelpe
virtualMachineConfiguration.networkDevices = [MacOSVirtualMachineConfigurationHelper
virtualMachineConfiguration.storageDevices = [MacOSVirtualMachineConfigurationHelper

virtualMachineConfiguration.pointingDevices = [MacOSVirtualMachineConfigurationHelpe
virtualMachineConfiguration.keyboards = [MacOSVirtualMachineConfigurationHelper.crea

try! virtualMachineConfiguration.validate()

if #available(macOS 14.0, *) {
```

```
    try! virtualMachineConfiguration.validateSaveRestoreSupport()
}

virtualMachine = VZVirtualMachine(configuration: virtualMachineConfiguration)
```

Inside the `createVirtualMachine` method, the app also creates a platform configuration for the VM. `VZMacPlatformConfiguration` configures important macOS-specific data that the macOS guest needs to run, including the specific `hardwareModel` that the image supports, as well as a `machineIdentifier` that uniquely identifies the current VM instance and differentiates it from any others.

Swift | Objective-C

```
let macPlatform = VZMacPlatformConfiguration()

let auxiliaryStorage = VZMacAuxiliaryStorage(contentsOf: auxiliaryStorageURL)
macPlatform.auxiliaryStorage = auxiliaryStorage

if !FileManager.default.fileExists(atPath: vmBundlePath) {
    fatalError("Missing Virtual Machine Bundle at \(vmBundlePath). Run InstallationT
}

// Retrieve the hardware model and save this value to disk
// during installation.
guard let hardwareModelData = try? Data(contentsOf: hardwareModelURL) else {
    fatalError("Failed to retrieve hardware model data.")
}

guard let hardwareModel = VZMacHardwareModel(dataRepresentation: hardwareModelData)
    fatalError("Failed to create hardware model.")
}

if !hardwareModel.isSupported {
    fatalError("The hardware model isn't supported on the current host")
}
macPlatform.hardwareModel = hardwareModel

// Retrieve the machine identifier and save this value to disk
// during installation.
guard let machineIdentifierData = try? Data(contentsOf: machineIdentifierURL) else {
    fatalError("Failed to retrieve machine identifier data.")
}
```

```
guard let machineIdentifier = VZMacMachineIdentifier(dataRepresentation: machineIder
    fatalError("Failed to create machine identifier.")
}
```

After creating the platform configuration, the app creates an instance of `VZVirtualMachineConfiguration` and adds video, virtual drives, and other devices to the system.

Swift | Objective-C

```
let virtualMachineConfiguration = VZVirtualMachineConfiguration()

virtualMachineConfiguration.platform = createMacPlatformConfiguration(macOSConfigura
virtualMachineConfiguration.cpuCount = MacOSVirtualMachineConfigurationHelper.comput
if virtualMachineConfiguration.cpuCount < macOSConfiguration.minimumSupportedCPUCour
    fatalError("CPUCount isn't supported by the macOS configuration.")
}

virtualMachineConfiguration.memorySize = MacOSVirtualMachineConfigurationHelper.comp
if virtualMachineConfiguration.memorySize < macOSConfiguration.minimumSupportedMemor
    fatalError("memorySize isn't supported by the macOS configuration.")
}

// Create a 128 GB disk image.
createDiskImage()

virtualMachineConfiguration.bootLoader = MacOSVirtualMachineConfigurationHelper.crea

virtualMachineConfiguration.audioDevices = [MacOSVirtualMachineConfigurationHelper.c
virtualMachineConfiguration.graphicsDevices = [MacOSVirtualMachineConfigurationHelpe
virtualMachineConfiguration.networkDevices = [MacOSVirtualMachineConfigurationHelper
virtualMachineConfiguration.storageDevices = [MacOSVirtualMachineConfigurationHelper

virtualMachineConfiguration.pointingDevices = [MacOSVirtualMachineConfigurationHelpe
virtualMachineConfiguration.keyboards = [MacOSVirtualMachineConfigurationHelper.crea

try! virtualMachineConfiguration.validate()

if #available(macOS 14.0, *) {
    try! virtualMachineConfiguration.validateSaveRestoreSupport()
}
```

```
virtualMachine = VZVirtualMachine(configuration: virtualMachineConfiguration)
virtualMachineResponder = MacOSVirtualMachineDelegate()
virtualMachine.delegate = virtualMachineResponder
```

The Virtualization framework checks the configuration to make sure it supports saving and restoring.

Start the VM

After building the configuration data for the VM, the sample app uses the `VZVirtualMachine` object to start the execution of the macOS guest operating system.

Before calling the `start(completionHandler:)` or `restoreMachineStateFrom(url:completionHandler:)` methods, the sample app configures a delegate object to receive messages about the state of the virtual machine. When the macOS guest operating system shuts down, the virtual machine calls the delegate's `guestDidStop(_:)` method. In response, the delegate method prints a message and exits the app. If the macOS guest stops for any reason other than a normal shutdown, the delegate prints an error message and the app exits.

Swift | Objective-C

```
DispatchQueue.main.async { [self] in
    createVirtualMachine()
    virtualMachineResponder = MacOSVirtualMachineDelegate()
    virtualMachine.delegate = virtualMachineResponder
    virtualMachineView.virtualMachine = virtualMachine
    virtualMachineView.capturesSystemKeys = true

    if #available(macOS 14.0, *) {
        // Configure the app to automatically respond to changes in the display size
        virtualMachineView.automaticallyReconfiguresDisplay = true
    }

    if #available(macOS 14.0, *) {
        let fileManager = FileManager.default
        if fileManager.fileExists(atPath: saveFileURL.path) {
            restoreVirtualMachine()
        } else {
            startVirtualMachine()
        }
    } else {
        startVirtualMachine()
```

```
    }
}
```

If the virtual machine was running when the sample app last exited, the app calls `restoreVirtualMachine` to restore the state. If the virtual machine was in a shutdown state, the app calls `startVirtualMachine` to reboot the machine. Both methods start the VM asynchronously in the background. The VM loads the system image and boots macOS. After macOS starts, the user interacts with a `VZVirtualMachineView` window that displays the macOS UI and handles keyboard and mouse input through a `VZMacGraphicsDeviceConfiguration` as though the user is interacting directly with the Mac hardware. The `VZVirtualMachineView` automatically resizes the virtual machine display when window size changes, and to capture system keys such, as the Globe key on a Mac keyboard.

The `startVirtualMachine` method calls the VM's `start(completionHandler:)` method.

Swift Objective-C

```swift
func startVirtualMachine() {
    virtualMachine.start(completionHandler: { (result) in
        if case let .failure(error) = result {
            fatalError("Virtual machine failed to start with \(error)")
        }
    })
}
```

Or, if the app previously had the VM save its state to `SaveFile.vzvmsave`, `restoreVirtualMachine` calls the VM's `restoreMachineStateFrom(url:completionHandler:)` and `resume(completionHandler:)` methods.

Swift Objective-C

```swift
func resumeVirtualMachine() {
    virtualMachine.resume(completionHandler: { (result) in
        if case let .failure(error) = result {
            fatalError("Virtual machine failed to resume with \(error)")
        }
    })
}

@available(macOS 14.0, *)
func restoreVirtualMachine() {
```

```
    virtualMachine.restoreMachineStateFrom(url: saveFileURL, completionHandler: { [s
        // Remove the saved file. Whether success or failure, the state no longer ma
        let fileManager = FileManager.default
        try! fileManager.removeItem(at: saveFileURL)

        if error == nil {
            self.resumeVirtualMachine()
        } else {
            self.startVirtualMachine()
        }
    })
}
```

If the restore fails, the framework causes the virtual machine to reboot. In either case, the framework deletes `SaveFile.vzvmsave` after restore completes because the VM disk no longer matches the state in the file.

Save the VM

When you close the sample app, it calls the VM's `pause(completionHandler:)` and `saveMachineStateTo(url:completionHandler:)` methods. This captures the runtime state of the VM to `SaveFile.vzvmsave`, which the app uses when calling `startOrRestoreVirtualMachine` to resume running the VM at the same point when you relaunch the sample app.

Swift | Objective-C

```
@available(macOS 14.0, *)
func saveVirtualMachine(completionHandler: @escaping () -> Void) {
    virtualMachine.saveMachineStateTo(url: saveFileURL, completionHandler: { (error)
        guard error == nil else {
            fatalError("Virtual machine failed to save with \(error!)")
        }

        completionHandler()
    })
}

@available(macOS 14.0, *)
func pauseAndSaveVirtualMachine(completionHandler: @escaping () -> Void) {
    virtualMachine.pause(completionHandler: { (result) in
        if case let .failure(error) = result {
            fatalError("Virtual machine failed to pause with \(error)")
```

```
        }

        self.saveVirtualMachine(completionHandler: completionHandler)
    })
}
```

The system defers app termination until the `saveMachineStateTo(url:completionHandler:)` method completes.

See Also

Virtual machine setup

- Running Linux in a Virtual Machine
  Run a Linux operating system on your Mac using the Virtualization framework.
- Running GUI Linux in a virtual machine on a Mac
  Install and run GUI Linux in a virtual machine using the Virtualization framework.
- Installing macOS on a Virtual Machine
  Download a macOS restore image and install it in a new VM.
- Creating and Running a Linux Virtual Machine
  Design and run custom Linux guests on Apple silicon or Intel-based Mac Computers.
- Virtualize macOS on a Mac
  Configure and run macOS guests on Apple silicon.
- Virtualize Linux on a Mac
  Configure and run Linux guests on Apple silicon and Intel-based Mac computers.
- Running Intel Binaries in Linux VMs with Rosetta
  Run x86_64 Linux binaries under ARM Linux on Apple silicon.
- Accelerating the performance of Rosetta
  Improve Rosetta performance by adding support for the total store ordering (TSO) memory model to your Linux kernel.