Sample Code

Running GUI Linux in a virtual machine on a Mac

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

Download

macOS 13.0+ | Xcode 14.0+

Overview

This sample code project demonstrates how to install and run GUI Linux virtual machines (VMs) on a Mac.

The Xcode project includes a single target, `GUILinuxVirtualMachineSampleApp`, which is a macOS app that installs a Linux distribution from an ISO image into a VM, and subsequently runs the installed Linux VM.

Download a Linux installation image

Before you run the sample program, you need to download an ISO installation image from a Linux distribution website. Some common Linux distributions include:

- Debian
- Fedora
- Ubuntu

Important

The Virtualization framework can run Linux VMs on a Mac with Apple silicon, and on an Intel-based Mac. The Linux ISO image you download must support the CPU architecture of your Mac. For a Mac with Apple silicon, download a Linux ISO image for ARM, which is usually indicated by `aarch64` or `arm64` in the image filename. For an Intel-based Mac, download a Linux ISO image for Intel-compatible CPUs, which is usually indicated by `x86_64` or `amd64` in the image filename.

Note

If you need to run Intel Linux binaries in ARM Linux on a Mac with Apple silicon, the Virtualization framework supports this capability using the Rosetta translation environment. For more information, see Running Intel Binaries in Linux VMs with Rosetta.

Configure the sample code project

Note

The default deployment target is macOS14, if you need to build for a different version of macOS you'll need to change the deployment target as appropriate.

1. Launch Xcode and open `GUILinuxVirtualMachineSampleApp.xcodeproj`.
2. Navigate to the Signing & Capabilities panel and select your team ID.
3. Build and run GUILinuxVirtualMachineSampleApp. The sample app starts the VM and configures a graphical view that you interact with. The Linux VM continues running until you shut it down from the guest OS, or when you quit the app.

   When you run the app for the first time, it displays a file picker so you can choose the Linux installation ISO image to use for installing your Linux VM. Navigate to the ISO image that you downloaded, select the file, and click Open. The VM boots into the OS installer, and the installer's user interface appears in the app's window. Follow the installation instructions. When the installation finishes, the Linux VM is ready to use.

   As part of the installation process, the Virtualization framework creates a `GUI Linux VM .bundle` package in your home directory. The sample app only supports running one VM at a time, however, the Virtualization framework supports running multiple VMs simultaneously. Running multiple VMs requires an app to manage the execution and artifacts of each individual VM.

The contents of the bundle represent the state of the Linux guest, and contain the following:

- `Disk.img` — The main disk image of the installed Linux OS.
- `MachineIdentifier` — The data representation of the `VZGenericMachineIdentifier` object.
- NVRAM — The EFI variable store.

Subsequent launches of GUILinuxVirtualMachineSampleApp run the installed Linux VM. To reinstall the VM, delete the `GUI Linux VM.bundle` package and run the app again.

Install GUI Linux from an ISO image

The sample app configures a `VZDiskImageStorageDeviceAttachment` object with the downloaded ISO image attached, and creates a `VZUSBMassStorageDeviceConfiguration` with it to emulate a USB thumb drive that's plugged in to the VM.

```
private func createUSBMassStorageDeviceConfiguration() -> VZUSBMassStorageDeviceCon
    guard let intallerDiskAttachment = try? VZDiskImageStorageDeviceAttachment(url:
        fatalError("Failed to create installer's disk attachment.")
    }

    return VZUSBMassStorageDeviceConfiguration(attachment: intallerDiskAttachment)
}
```

Set up the VM

The sample app uses a `VZVirtualMachineConfiguration` object to configure the basic characteristics of the VM, such as the CPU count, memory size, various device configurations, and a `VZEFIBootLoader` to load the Linux operating system into the VM.

```
let virtualMachineConfiguration = VZVirtualMachineConfiguration()

virtualMachineConfiguration.cpuCount = computeCPUCount()
virtualMachineConfiguration.memorySize = computeMemorySize()

let platform = VZGenericPlatformConfiguration()
let bootloader = VZEFIBootLoader()
let disksArray = NSMutableArray()

if needsInstall {
```

```
    // This is a fresh install: Create a new machine identifier and EFI variable sto
    // and configure a USB mass storage device to boot the ISO image.
    platform.machineIdentifier = createAndSaveMachineIdentifier()
    bootloader.variableStore = createEFIVariableStore()
    disksArray.add(createUSBMassStorageDeviceConfiguration())
} else {
    // The VM is booting from a disk image that already has the OS installed.
    // Retrieve the machine identifier and EFI variable store that were saved to
    // disk during installation.
    platform.machineIdentifier = retrieveMachineIdentifier()
    bootloader.variableStore = retrieveEFIVariableStore()
}

virtualMachineConfiguration.platform = platform
virtualMachineConfiguration.bootLoader = bootloader

disksArray.add(createBlockDeviceConfiguration())
guard let disks = disksArray as? [VZStorageDeviceConfiguration] else {
    fatalError("Invalid disksArray.")
}
virtualMachineConfiguration.storageDevices = disks

virtualMachineConfiguration.networkDevices = [createNetworkDeviceConfiguration()]
virtualMachineConfiguration.graphicsDevices = [createGraphicsDeviceConfiguration()]
virtualMachineConfiguration.audioDevices = [createInputAudioDeviceConfiguration(), c

virtualMachineConfiguration.keyboards = [VZUSBKeyboardConfiguration()]
virtualMachineConfiguration.pointingDevices = [VZUSBScreenCoordinatePointingDeviceCo
virtualMachineConfiguration.consoleDevices = [createSpiceAgentConsoleDeviceConfigura

try! virtualMachineConfiguration.validate()
virtualMachine = VZVirtualMachine(configuration: virtualMachineConfiguration)
```

Enable copy-and-paste support between the host and the guest

In macOS 13 and later, the Virtualization framework supports copy-and-paste of text and images between the Mac host and Linux guests through the SPICE agent clipboard-sharing capability. The example below shows the steps for configuring `VZVirtioConsoleDeviceConfiguration` and `VZSpiceAgentPortAttachment` to enable this capability:

```
private func createSpiceAgentConsoleDeviceConfiguration() -> VZVirtioConsoleDeviceCo
    let consoleDevice = VZVirtioConsoleDeviceConfiguration()

    let spiceAgentPort = VZVirtioConsolePortConfiguration()
    spiceAgentPort.name = VZSpiceAgentPortAttachment.spiceAgentPortName
    spiceAgentPort.attachment = VZSpiceAgentPortAttachment()
    consoleDevice.ports[0] = spiceAgentPort

    return consoleDevice
}
```

Important

To use the copy-and-paste capability in Linux, the user needs to install the `spice-vdagent` package, which is available through most Linux package managers. Developers need to communicate this requirement to users of their apps.

Start the VM

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

Before calling the VM's `start(completionHandler:)` method, the sample app configures a delegate object to receive messages about the state of the virtual machine. When the Linux operating system shuts down, the VM calls the delegate's `guestDidStop(_:)` method. In response, the delegate method prints a message and exits the sample.

```
self.virtualMachineView.virtualMachine = self.virtualMachine

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

self.virtualMachine.delegate = self
self.virtualMachine.start(completionHandler: { (result) in
    switch result {
    case let .failure(error):
        fatalError("Virtual machine failed to start with error: \(error)")

    default:
```

```
        print("Virtual machine successfully started.")
    }
})
```

The app sets the display to automatically resize when the window size changes.

See Also

Virtual machine setup

{} Running macOS in a virtual machine on Apple silicon

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

{} Running Linux in a Virtual Machine

Run a Linux operating system on your Mac 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.