PCET's
Pimpri
Chinchwad
University, Pune

Learn | Grow | Achieve

Checked By	Dr. Avinash Golande
Sign	
Date	
Grade	

EXPERIMENT NO . 03

- **Title:-** Study and Implementation of *libvirt* for Virtual Machine (VM) Managemente

- **Objective :-**
  - To understand libvirt as a virtualization management tool.
  - To learn how to create, configure, and manage VMs using libvirt.
  - To explore Python bindings of libvirt for programmatically controlling hypervisors.

- **Resources used :-** PC / Laptop with Linux OS (Ubuntu/Debian recommended) ,libvirt-daemon, qemu-kvm, virt-manager , Python 3 with libvirt-python module installed

- **Theory :-**

What is libvirt?
libvirt is basically the **"control tower"** for managing virtual machines, containers, and virtualization resources on a host system. Instead of directly talking to a specific hypervisor like QEMU/KVM, Xen, VirtualBox, or VMware ESXi, we interact with **libvirt**, which then translates our requests into the correct commands.

It includes:

- **Library:** libvirt.so
- **Daemon/Service:** libvirtd (or modular daemons in newer versions)
- **Tools:** virsh, virt-install, virt-manager

Main Components:

I. **libvirt API / library** – Interfaces for programming in C, Python, Go, Java.

II. **libvirtd service** – Handles communication with hypervisors (locally or remotely).

III. **CLI tools:**
   - virsh – Command line VM control
   - virt-install – For creating VMs

- virt-manager – GUI frontend

Capabilities of libvirt:

- Create and delete VMs
- Configure CPU, RAM, disks, and networks
- Attach/detach resources dynamically
- Live migrate VMs
- Take snapshots (disk + memory)
- Manage storage pools and virtual networks

- **Option A — Real libvirt + Python bindings (create / start / stop VMs)**

Prerequisites

I. Linux host with libvirtd running and qemu/kvm installed.

II. Python 3 and libvirt Python package:

bash

CopyEdit

sudo apt-get install -y libvirt-daemon-system qemu-kvm virtinst # Debian/Ubuntu example

pip install libvirt-python

I. qemu-img available to create QCOW2 images.

1. You run the script as a user with permission to talk to libvirt (often your user needs to be in libvirt/libvirt-qemu group or run via sudo).

Note: this script issues shell commands like qemu-img and manipulates domain XML — be careful on production systems.

I. What the script does
II. Create a backing QCOW2 image.
III. Create a minimal cloud-init ISO (optional) or use injected cloud-init user-data.
IV. Define a domain from an XML template and start the VM.
V. Provide functions to list, shutdown, destroy, undefine, snapshot.

```
#!/usr/bin/env python3
"""
Simple libvirt VM manager using libvirt Python bindings.
```

```
Provides: create_disk, define_domain, start, shutdown, destroy, undefine, list_domains
"""

import libvirt
import subprocess
import os
import tempfile
import time
from xml.etree import ElementTree as ET

LIBVIRT_URI = "qemu:///system"  # local system

def qemu_img_create(path, size_gb=10, backing=None):
    """Create a qcow2 image. If backing is provided, create a copy-on-write image."""
    cmd = ["qemu-img", "create", "-f", "qcow2"]
    if backing:
        cmd += ["-b", backing]
    cmd += [path, f"{size_gb}G"]
    subprocess.check_call(cmd)
    return path

def create_cloud_init_iso(hostname, ssh_authorized_key, iso_path):
    """Create a minimal cloud-init ISO with user-data/meta-data (for cloud images)."""
    tmpdir = tempfile.mkdtemp()
    user_data = f"""#cloud-config
hostname: {hostname}
ssh_authorized_keys:
  - {ssh_authorized_key}
"""
    with open(os.path.join(tmpdir, "user-data"), "w") as f:
```

```
        f.write(user_data)
    with open(os.path.join(tmpdir, "meta-data"), "w") as f:
        f.write("instance-id: iid-local01\nlocal-hostname: {}\n".format(hostname))
    # genisoimage or mkisofs
    cmd = ["genisoimage", "-output", iso_path, "-volid", "cidata", "-joliet", "-rock",
           os.path.join(tmpdir, "user-data"), os.path.join(tmpdir, "meta-data")]
    subprocess.check_call(cmd)
    return iso_path

def get_conn(uri=LIBVIRT_URI):
    return libvirt.open(uri)

def generate_domain_xml(name, disk_path, iso_path=None,
                        memory_mb=2048, vcpus=2, mac="52:54:00:12:34:56"):
    """
    Generate a basic domain XML for qemu+kvm.
    Adjust according to your environment (network, disk bus, etc.)
    """
    xml = f"""
<domain type='kvm'>
  <name>{name}</name>
  <memory unit='MiB'>{memory_mb}</memory>
  <vcpu placement='static'>{vcpus}</vcpu>
  <os>
    <type arch='x86_64' machine='pc'>hvm</type>
    <boot dev='hd'/>
  </os>
  <features>
    <acpi/>
    <apic/>
```

```
    <pae/>
  </features>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>destroy</on_crash>
  <devices>
    <emulator>/usr/bin/qemu-system-x86_64</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2'/>
      <source file='{disk_path}'/>
      <target dev='vda' bus='virtio'/>
    </disk>
"""
    if iso_path:
        xml += f"""
    <disk type='file' device='cdrom'>
      <driver name='qemu' type='raw'/>
      <source file='{iso_path}'/>
      <target dev='hdc' bus='ide'/>
      <readonly/>
    </disk>
"""
    xml += f"""
    <interface type='network'>
      <mac address='{mac}'/>
      <source network='default'/>
      <model type='virtio'/>
    </interface>
    <serial type='pty'><target port='0'/></serial>
```

```
    <console type='pty'><target type='serial' port='0'/></console>
    <input type='tablet' bus='usb'/>
    <graphics type='vnc' port='-1' listen='0.0.0.0'/>
    <memballoon model='virtio'/>
  </devices>
</domain>
"""
    return xml

def define_and_start(vm_name, disk_path, iso_path=None, memory_mb=2048, vcpus=2):
    conn = get_conn()
    xml = generate_domain_xml(vm_name, disk_path, iso_path, memory_mb, vcpus)
    try:
        dom = conn.defineXML(xml)
    except libvirt.libvirtError as e:
        print("Failed to define domain:", e)
        conn.close()
        raise
    print(f"Defined domain {vm_name}")
    dom.create()  # boot it
    print(f"Started domain {vm_name}")
    conn.close()
    return dom

def list_domains():
    conn = get_conn()
    ids = conn.listAllDomains()
    out = []
    for dom in ids:
        state, reason = dom.state()
```

```
        out.append((dom.name(), dom.ID(), state))
    conn.close()
    return out

def shutdown_domain(name):
    conn = get_conn()
    dom = conn.lookupByName(name)
    dom.shutdown()
    conn.close()

def destroy_and_undefine(name):
    conn = get_conn()
    dom = conn.lookupByName(name)
    if dom.isActive():
        dom.destroy()
    dom.undefine()
    conn.close()

# Example usage if run as script
if __name__ == "__main__":
    # PARAMETERS - change as needed
    vm_name = "demo-vm-1"
    disk_file = f"/var/lib/libvirt/images/{vm_name}.qcow2"
    base_image = "/var/lib/libvirt/images/ubuntu-22.04-server-cloudimg-amd64.img"  # must exist
    iso_path = f"/var/lib/libvirt/images/{vm_name}-cidata.iso"
    ssh_pubkey = "ssh-rsa AAAA...yourkey... user@example"
    # create disk (COW) backed by base image
    if not os.path.exists(disk_file):
        print("Creating qcow2 disk...")
        qemu_img_create(disk_file, size_gb=10, backing=base_image)
```

```
# create cloud-init ISO
if not os.path.exists(iso_path):
    create_cloud_init_iso(vm_name, ssh_pubkey, iso_path)
# define and start
define_and_start(vm_name, disk_file, iso_path, memory_mb=2048, vcpus=2)
print("VMs:", list_domains())
```

- Option B — VM Scheduling Simulator (pure Python)

If you want to *simulate scheduling* (pack VMs onto hosts using heuristics), here's a portable simulator you can run locally. It models hosts and VMs (CPU cores and RAM) and provides two placement heuristics: First Fit and Best Fit by remaining capacity.

Save as vm_scheduler_sim.py and run with Python 3.

```
#!/usr/bin/env python3
"""
VM scheduling simulator.
Hosts have capacities: cpu_cores, ram_gb
VMs request: cpu, ram
Heuristics: first-fit, best-fit (by residual RAM)
"""

from dataclasses import dataclass, field
from typing import List, Tuple
import random
import pprint

@dataclass
class VM:
    id: str
    cpu: int
    ram: int  # GB
```

```
@dataclass
class Host:
    id: str
    total_cpu: int
    total_ram: int
    used_cpu: int = 0
    used_ram: int = 0
    vms: List[VM] = field(default_factory=list)

    @property
    def free_cpu(self):
        return self.total_cpu - self.used_cpu

    @property
    def free_ram(self):
        return self.total_ram - self.used_ram

    def can_host(self, vm: VM) -> bool:
        return (self.free_cpu >= vm.cpu) and (self.free_ram >= vm.ram)

    def place_vm(self, vm: VM):
        if not self.can_host(vm):
            raise ValueError(f"Host {self.id} cannot host VM {vm.id}")
        self.used_cpu += vm.cpu
        self.used_ram += vm.ram
        self.vms.append(vm)

def schedule_first_fit(hosts: List[Host], vms: List[VM]) -> Tuple[List[Host], List[VM]]:
    unplaced = []
    for vm in vms:
```

```
        placed = False
        for host in hosts:
            if host.can_host(vm):
                host.place_vm(vm)
                placed = True
                break
        if not placed:
            unplaced.append(vm)
    return hosts, unplaced

def schedule_best_fit_ram(hosts: List[Host], vms: List[VM]) -> Tuple[List[Host], List[VM]]:
    unplaced = []
    for vm in vms:
        # find host with smallest free_ram that can still fit vm (best-fit)
        candidates = [h for h in hosts if h.can_host(vm)]
        if not candidates:
            unplaced.append(vm)
            continue
        best = min(candidates, key=lambda h: h.free_ram - vm.ram)
        best.place_vm(vm)
    return hosts, unplaced

def random_test():
    hosts = [Host(f"host{i}", total_cpu=32, total_ram=128) for i in range(4)]
    # create a set of VMs with random sizes
    vms = []
    for i in range(20):
        cpu = random.choice([1,2,4,8])
        ram = random.choice([1,2,4,8,16])
        vms.append(VM(f"vm{i}", cpu, ram))
```

```
    print("Scheduling with First-Fit")
    import copy
    h1 = [copy.deepcopy(h) for h in hosts]
    schedule_first_fit(h1, vms)
    for h in h1:
        print(h.id, "used_cpu", h.used_cpu, "used_ram", h.used_ram, "vms", len(h.vms))
    print()
    print("Scheduling with Best-Fit (RAM)")
    h2 = [copy.deepcopy(h) for h in hosts]
    schedule_best_fit_ram(h2, vms)
    for h in h2:
        print(h.id, "used_cpu", h.used_cpu, "used_ram", h.used_ram, "vms", len(h.vms))


if __name__ == "__main__":
    random_test()
```

- Example workflow: use both together
    I. Design VM descriptors (CPU, RAM, image, cloud-init key, name).
    II. Run simulator to decide which host each VM should land on (use the simulator heuristics).
    III. Write a small orchestration script that takes the simulator's placements and uses the libvirt script (Option A) to connect to each host's libvirt URI and define/start the VM there:
    - For remote hosts: qemu+ssh://user@host/system can be used as LIBVIRT_URI in Option A.
        a. Use SSH keys for passwordless remote libvirt access.

Conclusion:

Successfully studied and implemented **libvirt** as a virtualization management framework.

- Understood its architecture (library, daemon, CLI tools).
- Learned real VM management with libvirt Python bindings.
- Explored VM scheduling simulator to test placement strategies.