# VM Setup Guide for Polaris Subnet

## Introduction

This document outlines the process, requirements, and costs associated with migrating the Polaris subnet from a container-based architecture to a virtual machine (VM) based solution. It includes technical specifications, implementation steps, cost analysis, and recommendations.

## Container vs. VM Comparison

| Aspect | Containers (Current) | Virtual Machines (Proposed) |
| --- | --- | --- |
| Resource Usage | Lightweight, shared kernel | Heavyweight, full OS per instance |
| Isolation | Process-level isolation | Hardware-level isolation |
| Startup Time | Seconds | Minutes |
| Density | High (dozens per host) | Low (5-10 per host) |
| Flexibility | Limited to host OS family | Any OS supported by hypervisor |
| Security | Good, but shared kernel vulnerabilities | Strong isolation between VMs |
| Cost | Low resource overhead | High resource requirement |

## Hardware Requirements

### Minimum Server Specifications (Per Host)

For a production environment hosting 10-20 simultaneous development VMs:

* **CPU**: 16+ cores (32+ threads recommended)
* **RAM**: 64GB+ (128GB recommended)
* **Storage**: 1TB+ SSD (NVMe preferred)
* **Network**: 1Gbps minimum (10Gbps recommended)

### Client Requirements

* **CPU**: Support for hardware virtualization (Intel VT-x/AMD-V)
* **OS**: Linux with kernel 5.0+ (for optimal KVM support)

## Software Requirements

### Core Components

1. **Hypervisor**:
   * KVM/QEMU (recommended for Linux)
   * VMware ESXi (commercial alternative)
   * Hyper-V (for Windows Server environments)
2. **Management Layer**:
   * libvirt + libvirt-python
   * OpenStack (for larger deployments)
   * VMware vCenter (for ESXi)
3. **Programming Dependencies**:
   * Python 3.8+
   * libvirt-python
   * paramiko (SSH library)
   * pyvmomi (for VMware)
4. **VM Image Management**:
   * qemu-img
   * cloud-init for VM customization

## Implementation Steps

### 1. Hypervisor Setup

# Install KVM and libvirt on Ubuntu/Debian  
sudo apt update  
sudo apt install -y qemu-kvm libvirt-daemon-system libvirt-clients bridge-utils virtinst  
  
# Enable and start libvirt service  
sudo systemctl enable libvirtd  
sudo systemctl start libvirtd  
  
# Add current user to libvirt group  
sudo usermod -aG libvirt $USER  
sudo usermod -aG kvm $USER

### 2. Create Base VM Template

# Download Ubuntu Server image  
wget https://releases.ubuntu.com/22.04/ubuntu-22.04-live-server-amd64.iso  
  
# Create base VM image  
sudo virt-install --name polaris-base-template \  
 --memory 2048 \  
 --vcpus 2 \  
 --disk size=20 \  
 --network bridge=virbr0 \  
 --os-variant ubuntu22.04 \  
 --cdrom ubuntu-22.04-live-server-amd64.iso  
  
# After installation, configure the VM with necessary dev tools  
# Then shut down the VM and create a template  
sudo virt-sysprep -d polaris-base-template --enable customize --firstboot-command \  
 'apt-get update && apt-get install -y python3-pip git openssh-server'

### 3. Python Code Implementation

Create vm\_manager.py in the /home/tang/polaris\_go/polaris-subnet/src/ directory:

import libvirt  
import uuid  
import logging  
import time  
import xml.etree.ElementTree as ET  
from pathlib import Path  
  
class VMManager:  
 def \_\_init\_\_(self):  
 """Initialize connection to hypervisor."""  
 self.logger = logging.getLogger(\_\_name\_\_)  
 self.conn = libvirt.open('qemu:///system')  
 if not self.conn:  
 raise Exception("Failed to connect to hypervisor")  
   
 self.base\_template = "polaris-base-template"  
 self.vm\_storage\_path = "/var/lib/libvirt/images/"  
   
 def create\_vm(self, name=None):  
 """Create a new VM from the base template."""  
 if not name:  
 name = f"polaris-dev-{str(uuid.uuid4())[:8]}"  
   
 # Clone from base template  
 source\_dom = self.conn.lookupByName(self.base\_template)  
 source\_xml = source\_dom.XMLDesc(0)  
   
 # Modify XML to create new VM  
 tree = ET.fromstring(source\_xml)  
 tree.find('.//name').text = name  
   
 # Update UUID  
 new\_uuid\_elem = tree.find('.//uuid')  
 new\_uuid\_elem.text = str(uuid.uuid4())  
   
 # Update disk paths  
 disk\_elem = tree.find('.//disk[@device="disk"]')  
 source\_elem = disk\_elem.find('.//source')  
 old\_path = source\_elem.get('file')  
 new\_path = f"{self.vm\_storage\_path}{name}.qcow2"  
 source\_elem.set('file', new\_path)  
   
 # Create disk clone  
 import subprocess  
 subprocess.run([  
 'qemu-img', 'create', '-f', 'qcow2',   
 '-b', old\_path, new\_path  
 ])  
   
 # Define and start new VM  
 new\_xml = ET.tostring(tree).decode()  
 new\_dom = self.conn.defineXML(new\_xml)  
 new\_dom.create()  
   
 # Wait for VM to get IP address  
 max\_wait = 120 # seconds  
 start\_time = time.time()  
 ip\_address = None  
   
 while time.time() - start\_time < max\_wait:  
 if new\_dom.isActive():  
 try:  
 ifaces = new\_dom.interfaceAddresses(libvirt.VIR\_DOMAIN\_INTERFACE\_ADDRESSES\_SRC\_LEASE)  
 for \_, iface\_data in ifaces.items():  
 for addr in iface\_data['addrs']:  
 if addr['type'] == libvirt.VIR\_IP\_ADDR\_TYPE\_IPV4:  
 ip\_address = addr['addr']  
 break  
 except libvirt.libvirtError:  
 pass  
   
 if ip\_address:  
 break  
   
 time.sleep(5)  
   
 if not ip\_address:  
 self.logger.error(f"Could not determine IP address for VM {name}")  
 return None  
   
 return {  
 "vm\_id": name,  
 "ip\_address": ip\_address,  
 "ssh\_port": 22  
 }  
   
 def delete\_vm(self, vm\_id):  
 """Delete a VM."""  
 try:  
 dom = self.conn.lookupByName(vm\_id)  
 if dom.isActive():  
 dom.destroy() # Force shutdown  
 dom.undefine()  
   
 # Remove disk  
 disk\_path = f"{self.vm\_storage\_path}{vm\_id}.qcow2"  
 import os  
 if os.path.exists(disk\_path):  
 os.remove(disk\_path)  
   
 return True  
 except libvirt.libvirtError as e:  
 self.logger.error(f"Failed to delete VM {vm\_id}: {str(e)}")  
 return False

### 4. Update Compute Subnet API

Modify /home/tang/polaris\_go/polaris-subnet/compute\_subnet/src/services/container.py to use VMs instead of containers.

### 5. Update Installation Scripts

Modify installation scripts to include hypervisor setup and VM template creation.

## Cost Analysis

### Hardware Costs

| Component | Containers (Current) | VMs (Proposed) | Difference |
| --- | --- | --- | --- |
| CPU cores | 4-8 cores | 16-32 cores | 4x increase |
| RAM | 16-32GB | 64-128GB | 4x increase |
| Storage | 250GB-500GB | 1TB-2TB | 4x increase |
| Server Hardware | $1,000-$2,000 | $4,000-$8,000 | 4x increase |

### Software Costs

| Component | Containers (Current) | VMs (Open Source) | VMs (Commercial) |
| --- | --- | --- | --- |
| Core Platform | Docker (Free) | KVM/QEMU (Free) | VMware ESXi (  $$$) | | Management | Docker Compose (Free) | libvirt (Free) | vCenter ($$  $$) |
| Total Software | $0 | $0 | $1,000-$5,000+ |

### Cloud Costs (Monthly, if hosted in cloud)

| Resource | Containers | VMs | Difference |
| --- | --- | --- | --- |
| 10 instances | $100-$200 | $500-$1,000 | 5x increase |
| 50 instances | $500-$1,000 | $2,500-$5,000 | 5x increase |
| 100 instances | $1,000-$2,000 | $5,000-$10,000 | 5x increase |

### Development Costs

* **Engineering Time**: 80-160 hours to refactor codebase
* **Testing**: 40-80 hours for testing and validation
* **Documentation**: 20-40 hours for documentation updates
* **Total Dev Cost**: Approximately $14,000-$28,000 (at $100/hr)

### Performance Costs

* **Provisioning Time**: Containers (seconds) vs VMs (minutes)
* **Resource Utilization**: 3-5x more hardware resources needed for VMs
* **User Capacity**: 75-80% reduction in total users per host

## Recommendations

### Recommendation: Continue Using Containers

**I recommend against migrating to VMs for the following reasons:**

1. **Cost Efficiency**: Containers provide a 3-5x cost advantage over VMs in both hardware and operational expenses.
2. **Performance**: Containers offer faster startup times and lower latency, which is crucial for development environments.
3. **Scalability**: The current container architecture can scale to support more users without significant hardware investments.
4. **Development Effort**: The engineering effort to refactor the system would be substantial and likely not deliver enough benefits to justify the cost.

### Alternative Improvements to Consider

Instead of a full VM migration, consider these container improvements:

1. **Enhanced Isolation**: Implement stronger container isolation using features like:
   * Seccomp profiles
   * AppArmor/SELinux policies
   * User namespace isolation
   * Resource limits (cgroups)
2. **Hybrid Approaches**: For specific use cases requiring stronger isolation:
   * **Kata Containers**: VM-like security with container-like management
   * **gVisor**: Container runtime with an additional security layer
   * **Firecracker MicroVMs**: Lightweight VMs with fast startup times
3. **Storage Optimization**: Improve data persistence with named volumes, bind mounts, and backup strategies.
4. **Networking Enhancements**: Implement network policies for better security isolation between containers.

### When VMs Would Make Sense

VMs would only be justified if:

1. **Regulatory Requirements**: Your compliance framework specifically requires hardware-level isolation
2. **Operating System Diversity**: Users need to run non-Linux operating systems
3. **Kernel Modifications**: Users need to modify kernel parameters or load custom kernel modules
4. **Legacy Applications**: Supporting applications that cannot be containerized

## Conclusion

While migrating to VMs is technically feasible, the significant cost increases (3-5x) and performance penalties make it difficult to recommend for the Polaris subnet. The current container-based architecture provides a more cost-effective and efficient solution for development environments.

If specific security or isolation concerns exist, consider the hybrid approaches mentioned above before a full VM migration.