## Promts:

## Diagnosing high CPU/memory/disk usage in a Linux VM hosted on vSphere. so i need promts or solution or commands to check CPU/memory/disk usage in a linus machine i know the commands like free -h or df -h but for this diagnosing give me the excaxt and right commands to check this in a linux VM hsted on Vsphere

**1. CPU Diagnosis**

**Linux VM (inside the guest OS):**

# Real-time CPU usage per process

top

# More detailed, sorted by CPU

htop # (if installed)

# CPU usage summary

mpstat -P ALL 1 5

# Process stats with CPU usage

ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%cpu | head

**vSphere (ESXi/Host level):**

* In **vSphere Client**, check:
  + CPU Ready % → High value = VM waiting for CPU from host.
  + CPU Co-Stop % → For multi-vCPU VMs, imbalance issues.

### ****2. Memory Diagnosis****

**Linux VM:**

# Memory usage summary

free -h

# Top processes consuming memory

ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%mem | head

# Detailed memory usage by process

smem # (if available)

# Check swap usage

swapon –show

**vSphere:**

* Monitor in **vSphere Client**:
  + Ballooning
  + Swapping
  + Memory usage (%)

**3. Disk I/O Diagnosis**

**# Disk space usage**

**df -h**

**# I/O usage per device**

**iostat -xz 1 5**

**# Per-process I/O usage**

**iotop # (if installed)**

**# Check inode usage**

**df -i**

**vSphere:**

* Check datastore latency (read/write) in **vSphere performance charts**.
* Look for **high latency (>20 ms)** which indicates a storage bottleneck.

**4. Network Check (Optional but useful)**

**# Bandwidth usage**

**sar -n DEV 1 5**

**# Errors or dropped packets**

**netstat -i**

## Prompts:

## Automating VM snapshot creation, backup, and rollback using PowerCLI or Python + pyVmomi. so we have to create snapshot and taking backup and restore and rollback the data , write the steps for this with accurate information

## 🔹 Steps for Automating Snapshot, Backup, Restore, and Rollback

### 1. Prerequisites

* Access to **vCenter Server** or ESXi host.
* Credentials with permissions to create/delete snapshots.
* For PowerCLI → Install VMware PowerCLI (Install-Module VMware.PowerCLI).
* For Python → Install pyVmomi (pip install pyvmomi).

### 2. Snapshot Workflow (General)

1. **Connect** to vCenter/ESXi.
2. **Locate the VM** by name or ID.
3. **Create a snapshot** (with name, description, memory state, and quiesce options).
4. **Backup the VM** (optional: export VM or copy VMDK files).
5. **Rollback/restore**:
   * Either revert to the snapshot (rollback).
   * Or restore the VM from the backup copy.
6. **Clean up old snapshots** (important to avoid datastore bloat).

## Prompts:

## Automating VM snapshot creation, backup, and rollback using PowerCLI or Python + pyVmomi. so we have to create snapshot and taking backup and restore and rollback the data , write the steps for this with accurate information

## 🔹 Using **PowerCLI**

### Create Snapshot

# Connect to vCenter

Connect-VIServer -Server vcenter.example.com -User admin -Password 'yourpassword'

# Create snapshot

New-Snapshot -VM "MyVM" -Name "PrePatchSnapshot" -Description "Before patching" -Quiesce -Memory:$true

### Revert (Rollback) to Snapshot

# Get snapshot

$snap = Get-Snapshot -VM "MyVM" -Name "PrePatchSnapshot"

# Revert to snapshot

Set-VM -VM "MyVM" -Snapshot $snap -Confirm:$false

### Remove Snapshot

# Delete snapshot after successful update

Remove-Snapshot -Snapshot $snap -Confirm:$false

### Backup VM (Export OVF/OVA)

Export-VApp -VM "MyVM" -Destination "C:\VM-Backups"

## 🔹 Using **Python + pyVmomi**

### Connect to vCenter

from pyVim.connect import SmartConnect, Disconnect

import ssl

context = ssl.\_create\_unverified\_context()

si = SmartConnect(host="vcenter.example.com", user="administrator@vsphere.local", pwd="yourpassword", sslContext=context)

### Create Snapshot

from pyVmomi import vim

def create\_snapshot(vm, name, description, memory=False, quiesce=False):

task = vm.CreateSnapshot\_Task(name=name, description=description, memory=memory, quiesce=quiesce)

return task

# Example: Find VM and snapshot it

content = si.RetrieveContent()

for datacenter in content.rootFolder.childEntity:

vm = datacenter.vmFolder.childEntity[0] # Example: pick first VM

create\_snapshot(vm, "PrePatchSnapshot", "Before patching", True, True)

### Revert to Snapshot

def revert\_to\_snapshot(vm, snapshot\_name):

snap\_info = vm.snapshot

tree = snap\_info.rootSnapshotList

snapshot = None

def find\_snap(tree, name):

for node in tree:

if node.name == name:

return node.snapshot

snap = find\_snap(node.childSnapshotList, name)

if snap:

return snap

return None

snapshot = find\_snap(tree, snapshot\_name)

if snapshot:

task = snapshot.RevertToSnapshot\_Task()

return task

else:

print("Snapshot not found")

revert\_to\_snapshot(vm, "PrePatchSnapshot")

### Remove Snapshot

def remove\_snapshot(vm, snapshot\_name):

snap\_info = vm.snapshot

tree = snap\_info.rootSnapshotList

snapshot = None

def find\_snap(tree, name):

for node in tree:

if node.name == name:

return node.snapshot

snap = find\_snap(node.childSnapshotList, name)

if snap:

return snap

return None

snapshot = find\_snap(tree, snapshot\_name)

if snapshot:

task = snapshot.RemoveSnapshot\_Task(removeChildren=False)

return task

else:

print("Snapshot not found")

## 🔹 Overview

**VMware Tools** is a suite installed inside the guest VM that enables:

* Accurate guest OS metrics (CPU, memory, disk, network)
* Heartbeat monitoring
* Quiesced snapshots
* Time synchronization

With VMware Tools, you can monitor **real-time metrics** from the guest OS, which are more precise than the host-level metrics.

Metrics accessible include:

* CPU usage
* Memory usage (used, ballooned, swapped)
* Disk usage and latency
* Network usage
* Guest OS uptime and heartbeat

## Prompts:

## Monitoring VM performance via VMware Tools integration (guest metrics). i am monitoring the vm performance with vmware tools so how to monitoe the vn performance give some info and commands ?

## 🔹 Methods to Monitor VM Performance

### 1. ****Using vSphere Client (GUI)****

1. Go to **vSphere Client → VM → Monitor → Performance → Advanced → Charts → Guest OS**.
2. Select metrics like:
   * **CPU**: Usage %, Ready time
   * **Memory**: Consumed, Ballooned, Swapped
   * **Disk**: Read/Write throughput, latency
   * **Network**: Packets sent/received, throughput

### 2. ****Using PowerCLI (CLI)****

#### Connect to vCenter

Connect-VIServer -Server vcenter.example.com -User admin -Password 'yourpassword'

#### Get Guest Metrics

# Get VM object

$vm = Get-VM -Name "MyVM"

# Get CPU and Memory guest metrics

$vm | Get-VMGuest | Select-Object VMName, HostName, State, ToolsVersion, MemoryUsage, CPUUsage

#### Get Disk and Network Metrics

$vm | Get-VMGuestNetworkInterface | Select-Object Name, IPAddress, MacAddress, Status

$vm | Get-VMGuestDisk | Select-Object Path, FreeSpaceGB, CapacityGB

PowerCLI uses **VMware Tools** inside the guest to fetch these metrics. If VMware Tools isn’t running, guest metrics won’t be available.

### 3. ****Using Python + pyVmomi****

You can fetch **guest metrics via guestOperationsManager**.

from pyVim.connect import SmartConnect, Disconnect

import ssl

context = ssl.\_create\_unverified\_context()

si = SmartConnect(host="vcenter.example.com", user="administrator@vsphere.local", pwd="password", sslContext=context)

content = si.RetrieveContent()

# Select VM

vm = None

for dc in content.rootFolder.childEntity:

for v in dc.vmFolder.childEntity:

if v.name == "MyVM":

vm = v

break

# Get guest metrics

if vm.guest.toolsRunningStatus == 'guestToolsRunning':

print("VMware Tools is running")

print(f"CPU Usage: {vm.summary.quickStats.overallCpuUsage} MHz")

print(f"Memory Usage: {vm.summary.quickStats.guestMemoryUsage} MB")

print(f"Uptime: {vm.guest.guestHeartbeatStatus}")

else:

print("VMware Tools not running, metrics unavailable")

#### Notes:

* overallCpuUsage → Host-side CPU usage in MHz
* guestMemoryUsage → Memory used inside the guest in MB
* guestHeartbeatStatus → Shows guest OS heartbeat (green/yellow/red)

### 4. ****VMware ESXi CLI (esxcli)****

You can also check VM metrics from the ESXi host:

# List running VMs

vim-cmd vmsvc/getallvms

# Get VM summary

vim-cmd vmsvc/get.summary <vmid>

# Check CPU and memory usage

vim-cmd vmsvc/get.performance <vmid>

Metrics here are mostly host-side. For guest OS-specific info, VMware Tools metrics are more accurate.

**Monitoring VM performance via VMware Tools integration (guest metrics).**

* Always **keep VMware Tools updated**.
* Use **PowerCLI or vSphere API** for automated monitoring.
* Combine **host-level (ESXi)** and **guest-level (VMware Tools)** metrics for complete monitoring.
* Set up **alerts in vCenter** or **monitoring tools (e.g., vROps, Zabbix, Nagios)** using guest
* metrics.\

## 🔹 Methods to Monitor VM Performance

### 1. ****Using vSphere Client (GUI)****

1. Go to **vSphere Client → VM → Monitor → Performance → Advanced → Charts → Guest OS**.
2. Select metrics like:
   * **CPU**: Usage %, Ready time
   * **Memory**: Consumed, Ballooned, Swapped
   * **Disk**: Read/Write throughput, latency
   * **Network**: Packets sent/received, throughput

### 2. ****Using PowerCLI (CLI)****

#### Connect to vCenter

Connect-VIServer -Server vcenter.example.com -User admin -Password 'yourpassword'

#### Get Guest Metrics

# Get VM object

$vm = Get-VM -Name "MyVM"

# Get CPU and Memory guest metrics

$vm | Get-VMGuest | Select-Object VMName, HostName, State, ToolsVersion, MemoryUsage, CPUUsage

Get Disk and Network Metrics

$vm | Get-VMGuestNetworkInterface | Select-Object Name, IPAddress, MacAddress, Status

$vm | Get-VMGuestDisk | Select-Object Path, FreeSpaceGB, CapacityGB

### 3. ****Using Python + pyVmomi****

You can fetch **guest metrics via guestOperationsManager**.

from pyVim.connect import SmartConnect, Disconnect

import ssl

context = ssl.\_create\_unverified\_context()

si = SmartConnect(host="vcenter.example.com", user="administrator@vsphere.local", pwd="password", sslContext=context)

content = si.RetrieveContent()

# Select VM

vm = None

for dc in content.rootFolder.childEntity:

for v in dc.vmFolder.childEntity:

if v.name == "MyVM":

vm = v

break

# Get guest metrics

if vm.guest.toolsRunningStatus == 'guestToolsRunning':

print("VMware Tools is running")

print(f"CPU Usage: {vm.summary.quickStats.overallCpuUsage} MHz")

print(f"Memory Usage: {vm.summary.quickStats.guestMemoryUsage} MB")

print(f"Uptime: {vm.guest.guestHeartbeatStatus}")

else:

print("VMware Tools not running, metrics unavailable")

#### Notes:

* overallCpuUsage → Host-side CPU usage in MHz
* guestMemoryUsage → Memory used inside the guest in MB
* guestHeartbeatStatus → Shows guest OS heartbeat (green/yellow/red)

### 4. ****VMware ESXi CLI (esxcli)****

You can also check VM metrics from the ESXi host:

# List running VMs

vim-cmd vmsvc/getallvms

# Get VM summary

vim-cmd vmsvc/get.summary <vmid>

# Check CPU and memory usage

vim-cmd vmsvc/get.performance <vmid>

## Prompts : Designing auto-recovery logic using AI + shell/Python/Ansible if the Linux VM becomes unresponsive. if the linux env becomes not responsive design the auto recovery logic for this

## 🔹 1. **Overview of Auto-Recovery Logic**

**Goal:** Detect unresponsive Linux VMs and recover them automatically without manual intervention.

**Steps:**

1. **Health Monitoring** – Check CPU, memory, disk, network, and service availability.
2. **Anomaly Detection (Optional AI Layer)** – Use AI/ML to predict failure before complete crash.
3. **Automated Remediation** – Run recovery steps if VM is unresponsive:
   * Restart unresponsive services
   * Clear high-memory/CPU processes
   * Reboot VM
   * Trigger snapshots or failover if needed
4. **Alerting & Logging** – Notify admins after recovery action.

## 🔹 2. **Monitoring VM Health**

**Shell/Python commands to detect unresponsiveness:**

* **Ping Check (Network)**

ping -c 3 <VM\_IP>

if [ $? -ne 0 ]; then

echo "VM unreachable"

fi

CPU/Memory Check

CPU=$(top -bn1 | grep "Cpu(s)" | awk '{print $2 + $4}')

MEM=$(free | grep Mem | awk '{print $3/$2 \* 100.0}')

if (( $(echo "$CPU > 90" | bc -l) )); then

echo "CPU high"

fi

if (( $(echo "$MEM > 90" | bc -l) )); then

echo "Memory high"

fi

Service Check

systemctl is-active sshd || echo "sshd service down"

1. **Recovery Actions**

Shell/Python/Ansible Steps:

sudo systemctl restart <service\_name>

Python example:

import subprocess

services = ["sshd", "nginx"]

for s in services:

subprocess.run(["systemctl", "restart", s])

Kill High Resource Processes

ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%mem | head -n 5

# Kill top memory process

kill -9 <PID>

. Reboot VM if still unresponsive

sudo reboot

## 🔹 4. **Automated Workflow Design**

1. **Monitor VM**
   * Ping check
   * CPU/Memory thresholds
   * Service health
2. **AI Layer (Optional)**
   * Predict unresponsiveness
3. **Trigger Recovery**
   * Restart services → Kill problematic processes → Reboot VM
4. **Logging & Alerting**
   * Send email/SMS/slack notification with action taken.
5. **Post-Recovery Check**
   * Verify services are back online
   * Collect metrics for future AI training

Example Python Script Skeleton

import subprocess

import smtplib

def check\_ping(vm\_ip):

return subprocess.run(["ping", "-c", "3", vm\_ip]).returncode == 0

def check\_services(services):

down = []

for s in services:

result = subprocess.run(["systemctl", "is-active", s], capture\_output=True)

if b'inactive' in result.stdout:

down.append(s)

return down

def restart\_services(services):

for s in services:

subprocess.run(["systemctl", "restart", s])

def reboot\_vm():

subprocess.run(["reboot"])

# Main logic

vm\_ip = "192.168.1.10"

services = ["sshd", "nginx"]

if not check\_ping(vm\_ip):

print("VM unreachable, triggering recovery")

down\_services = check\_services(services)

if down\_services:

restart\_services(down\_services)

reboot\_vm()

## Prompts : Building a VM provisioning workflow using cloud-init + vSphere templates. i am building a Vm provisioning workflow using cloud -init vm Sphere templates , so we need accurate promts and the solutions and commands for this

## 🔹 1. **Overview**

**Goal:** Automate provisioning of Linux VMs in vSphere using **templates** and **cloud-init** to handle initial configuration (hostname, network, users, packages, SSH keys).

**Components:**

1. **vSphere VM Template** – A base Linux VM with cloud-init installed.
2. **Cloud-init config** – YAML file for guest customization.
3. **Provisioning workflow** – PowerCLI, Python (pyVmomi), or Terraform to clone VMs from templates.

## 🔹 2. **Prepare vSphere Template with Cloud-Init**

### Steps:

1. Deploy a **base Linux VM** (Ubuntu/CentOS/RHEL).
2. Install **cloud-init**:

# Ubuntu/Debian

sudo apt update

sudo apt install cloud-init -y

# RHEL/CentOS

sudo yum install cloud-init -y

Configure cloud-init datasource for vSphere:

sudo vi /etc/cloud/cloud.cfg.d/99-vmware.cfg

# Example content

datasource\_list: [VMware, None]

Disable unnecessary services, clean machine ID:

sudo cloud-init clean --logs

sudo systemctl enable cloud-init

## 🔹 3. **Create Cloud-Init User Data File**

**cloud-init YAML example (user-data.yaml)**:

#cloud-config

hostname: myvm-01

fqdn: myvm-01.example.com

manage\_etc\_hosts: true

users:

- name: admin

sudo: ALL=(ALL) NOPASSWD:ALL

ssh\_authorized\_keys:

- ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQD...

packages:

- htop

- git

- vim

runcmd:

- echo "Provisioned by cloud-init" > /etc/motd

- systemctl restart ssh

#cloud-config

hostname: myvm-01

fqdn: myvm-01.example.com

manage\_etc\_hosts: true

users:

- name: admin

sudo: ALL=(ALL) NOPASSWD:ALL

ssh\_authorized\_keys:

- ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQD...

packages:

- htop

- git

- vim

runcmd:

- echo "Provisioned by cloud-init" > /etc/motd

- systemctl restart ssh

🔹 4. **VM Provisioning Workflow**

# Connect to vCenter

Connect-VIServer -Server vcenter.example.com -User admin -Password 'password'

# Clone VM from template

$vm = Get-Template -Name "Ubuntu-Template" |

New-VM -Name "MyVM-01" -VMHost "esxi-host1" -Datastore "Datastore1"

# Apply customization spec for network/cloud-init if needed

# Optionally, add cloud-init ISO as CD-ROM to VM

New-Iso -Path "C:\cloud-init\user-data.iso" -VM $vm

# Power on VM

Start-VM $vm

B. **Using Python + pyVmomi**

from pyVim.connect import SmartConnect, Disconnect

from pyVmomi import vim

import ssl

# Connect to vCenter

context = ssl.\_create\_unverified\_context()

si = SmartConnect(host="vcenter.example.com", user="administrator@vsphere.local", pwd="password", sslContext=context)

content = si.RetrieveContent()

# Find template

template = None

for dc in content.rootFolder.childEntity:

for vm in dc.vmFolder.childEntity:

if vm.name == "Ubuntu-Template":

template = vm

# Clone VM

relocate\_spec = vim.vm.RelocateSpec(pool=template.resourcePool, datastore=template.datastore[0])

clone\_spec = vim.vm.CloneSpec(location=relocate\_spec, powerOn=True)

task = template.Clone(folder=template.parent, name="MyVM-01", spec=clone\_spec)

**Attach cloud-init ISO:**

* Generate ISO with user-data/meta-data using cloud-localds:

### C. ****Workflow Summary****

1. Prepare **cloud-init ready template VM**.
2. Generate **cloud-init user-data (YAML)** and optional meta-data.
3. Convert user-data to **ISO** and attach to VM at provisioning.
4. **Clone VM** from template using PowerCLI, pyVmomi, or Terraform.
5. **Power on VM** → cloud-init runs automatically → applies configuration.
6. **Verify VM** – hostname, users, network, installed packages.

“The intent behind each prompt is to obtain a clear solution or guidance for the tasks or questions I have.”