Virtual Instances and Server Consolidation with Fault Tolerance – Step-by-Step

---

Step 0 – Prerequisites

- Windows PC with **VMware Workstation Pro** installed (with WHP enabled if prompted).
- **Ubuntu 24.04 Desktop ISO** (ubuntu-24.04.2-desktop-amd64.iso).
- Minimum RAM: **8 GB recommended for 3 VMs** (2–4 GB per VM).
- Basic familiarity with **Ubuntu terminal**.

Step 1 – Install VMware Workstation

1. Run VMware-workstation-full-17.6.4-24832109.exe.
2. If prompted about WHP, check **“Install Windows Hypervisor Platform (WHP) automatically”** → Next.
3. Accept license → Next → Install → Finish.
4. Restart PC if required.

✅ VMware is now ready to create VMs.

Step 2 – Create Ubuntu Virtual Machines

2.1 Create First VM (fy1 / WebServer1)

1. File → New Virtual Machine → Typical → Next.
2. Select ISO: ubuntu-24.04.2-desktop-amd64.iso → Next.
3. Guest OS: Linux → Ubuntu 64-bit → Next.
4. VM Name: **fy1** → Next.
5. Disk: 20 GB, Store as single file → Next.
6. Customize Hardware:
    - Memory: 2048 MB (2 GB)
    - Processors: 2 cores
    - Network Adapter: Bridged → Next
7. Click **Finish** → **Power on VM** → Ubuntu installer starts.

2.2 Install Ubuntu on fy1

1. Select **Install Ubuntu** → Continue.
2. Keyboard layout → Next.
3. Installation type: **Erase disk and install Ubuntu** → Next.
4. Create user account: **student / password 1234** (example) → Install.

5. Wait ~10–15 mins → Reboot → remove ISO when prompted.

Step 3 – Configure Networking

1. Open terminal (Ctrl+Alt+T) → check IP:

```
ip addr show
```

- Look for `ens33` or `enp0s3` → note IP (e.g., `192.168.190.128`)

2. Test connectivity:

```
ping google.com
```

- If ping fails → VM settings → Network Adapter → Bridged → Replicate physical connection state.

Step 4 – Update Ubuntu

```
sudo apt update
sudo apt upgrade -y
```

Step 5 – Install Nginx Web Server

```
sudo apt install -y nginx
sudo systemctl enable --now nginx
```

- Test locally:

```
curl http://localhost
```

- Should show default Nginx page.

Step 6 – Customize Homepage

```
sudo nano /var/www/html/index.html
```

- Replace content with:

```
<h1>Hello from fy1</h1>
```

- Save: Ctrl+O → Enter, Exit: Ctrl+X
- Test:

```
curl http://localhost
```

Step 7 – Clone fy1 to fy2 and fy3

1. Shut down fy1 (optional).
2. Right-click fy1 → **Manage** → **Clone** → Full Clone.

3 . Name first clone: **fy2**, second clone: **fy3**.

4. Power on fy2 and fy3.

5. Update Nginx homepage for identification:

```
# On fy2
sudo nano /var/www/html/index.html
<h1>Hello from fy2</h1>

# On fy3
sudo nano /var/www/html/index.html
<h1>Hello from fy3</h1>
```

- Test each VM:

```
curl http://localhost
```

Step 8 – Install HAProxy on fy1 (Load Balancer)

```
sudo apt update
sudo apt install -y haproxy
sudo systemctl enable --now haproxy
```

Step 9 – Configure HAProxy

1. Edit config:

```
sudo nano /etc/haproxy/haproxy.cfg
```

2. Add at the end (replace with actual IPs of fy2/fy3):

```
frontend http_front
    bind *:80
    default_backend http_back

backend http_back
    balance roundrobin
    server fy2 192.168.190.129:80 check
    server fy3 192.168.190.130:80 check
```

3 . **Alternative:** Use VM names via /etc/hosts on fy1:

```
sudo nano /etc/hosts
192.168.190.129   fy2
192.168.190.130   fy3
```

Then in HAProxy config:

```
server fy2 fy2:80 check
server fy3 fy3:80 check
```

4. Restart HAProxy:

```
sudo systemctl restart haproxy
```

5. Test from host browser:

```
http://<fy1_IP>
```

- Refresh → pages from fy2 and fy3 alternate (round-robin).

```
# On both machines(optional)
sudo apt install apache2 -y
sudo systemctl restart apache2
```

Step 10 – Demonstrate Server Consolidation

1. Stop fy3 Nginx (simulate consolidation):

```
sudo systemctl stop nginx   # on fy3
```

2. Merge content into fy2 (optional demo):

```
echo "<h1>Merged Server: fy2 + fy3 content</h1>" | sudo tee
/var/www/html/index.html
```

3. Update HAProxy config to remove fy3 → restart:

```
sudo nano /etc/haproxy/haproxy.cfg
sudo systemctl restart haproxy
```

- Now all traffic goes to fy2 → demonstrates **resource optimization**.

Step 11 – Demonstrate Fault Tolerance

1. Stop Nginx on fy2 (simulate failure):

```
sudo systemctl stop nginx
```

- HAProxy detects fy2 down → stops sending traffic there.

2. Start Nginx again:

```
sudo systemctl start nginx
```

- Traffic resumes → demonstrates **fault-tolerant behavior**.

- Optional network diagram:

```
      Browser
          |
        fy1 (HAProxy)
       /          \
  fy2 (Nginx)     fy3 (Nginx)
```

Task Scheduling and Load Balancing in a Cloud Environment – Step-by-Step Guide

Step 0 – Prerequisites

- **VMware Workstation** with Ubuntu VMs: **fy1** (controller), **fy2** and **fy3** (workers).
- Internet connection (for package installation).
- RAM: 2–4 GB per VM.
- Basic familiarity with **Ubuntu terminal commands**.

> **Why:** Multiple VMs simulate a cloud environment for scheduling and load balancing tasks.

Part 1 – Set Up Passwordless SSH for Task Scheduling

Passwordless SSH allows automated tasks to run on workers from fy1 **without typing passwords**, enabling centralized task management.

Step 1 – Install and Start SSH on Worker VMs

On **fy2** and **fy3** terminals:

```
sudo apt update
```

```
sudo apt install -y openssh-server
sudo systemctl start ssh

sudo systemctl status ssh
```

> **Why:** SSH server must be running so fy1 can connect remotely to schedule tasks.

Step 2 – Generate SSH Key on Controller VM

On **fy1** terminal:

```
ssh-keygen -t rsa
```

- Press **Enter** for all prompts (no passphrase).

> **Why:** Creates a **public-private key pair**. The private key stays on fy1, public key will go to fy2/fy3 for authentication.

Step 3 – Copy Public Key to Workers

On **fy1** terminal:

```
ssh-copy-id student@192.168.190.129  # fy2

ssh-copy-id student@192.168.190.130  # fy3
```

- Type **yes** when prompted and enter the VM password once.

Why: Enables **passwordless SSH login** for automated cron tasks.

Step 4 – Test Passwordless SSH

On **fy1** terminal:

```
ssh student@192.168.190.129 'hostname'

ssh student@192.168.190.130 'hostname'
```

Expected: fy2 and fy3 respond with their hostnames without asking for a password.
Why: Confirms passwordless SSH is working, essential for automated task execution.

Step 5 – How Passwordless SSH Works (Conceptual)

1. **Initiation:** fy1 requests login using its key.

2. **Challenge:** fy2/fy3 encrypts a secret message using the public key.

3. **Response:** fy1 decrypts it using its private key.

4. **Verification:** If it matches, access is granted **without a password**.

Why: Required for automated task execution across multiple VMs.

Part 2 – Task Scheduling Using Cron

Step 6 – Open Crontab on Controller VM

```
crontab -e
```

- Select **nano** if prompted.

Step 7 – Schedule Cron Tasks

Add these lines to the bottom:

```
# Run every minute on fy2

* * * * * ssh student@192.168.190.129 'echo "Task executed on $(hostname) at $(date)" >> /tmp/cron_job.log'


# Run every minute on fy3

* * * * * ssh student@192.168.190.130 'echo "Task executed on $(hostname) at $(date)" >> /tmp/cron_job.log'
```

> **Why:** Demonstrates **distributed automated tasks** running every minute on worker VMs.

Step 8 – Verify Tasks

On **fy1** terminal:

```
ssh student@192.168.190.129 'cat /tmp/cron_job.log'

ssh student@192.168.190.130 'cat /tmp/cron_job.log'
```

> **Expected:** Logs show timestamps of task executions.
> **Why:** Confirms cron jobs run correctly on remote VMs.

Optional – Advanced Scheduling

Add tasks with **priority**:

```
# High priority every minute

* * * * * ssh student@192.168.190.129 'echo "High priority task $(date)" >> /tmp/priority_log.txt'


# Low priority every 5 minutes

*/5 * * * * ssh student@192.168.190.130 'echo "Low priority task $(date)" >> /tmp/priority_log.txt'
```

> **Why:** Demonstrates **priority-based scheduling**, simulating cloud workloads with different resource requirements.

Part 3 – Load Balancing with HAProxy

Step 9 – Prepare Web Servers

On **fy2** and **fy3**:

```
sudo apt install -y nginx
```

- Customize homepage content:

```
# fy2
echo "<h1>Hello from fy2</h1>" | sudo tee /var/www/html/index.html


# fy3
echo "<h1>Hello from fy3</h1>" | sudo tee /var/www/html/index.html
```

- Test locally:

```
curl http://localhost
```

> **Why:** Prepares web servers for HAProxy load balancing.

Step 10 – Install HAProxy on Controller VM (fy1)

```
sudo apt update
sudo apt install -y haproxy
sudo systemctl enable --now haproxy
```

Why: HAProxy acts as a **load balancer** distributing incoming HTTP requests to fy2/fy3.

Step 11 – Configure HAProxy

```
sudo nano /etc/haproxy/haproxy.cfg
```

Add at the end:

```
frontend http_front
    bind *:80
    default_backend http_back

backend http_back
    balance roundrobin
    server fy2 192.168.190.129:80 check
    server fy3 192.168.190.130:80 check
```

Why:

- `frontend`: listens on port 80.
- `backend`: distributes traffic **round-robin** between fy2 and fy3.
- `check`: automatically detects if a server is down.

Optional – Weighted Load Balancing

```
backend http_back
    balance roundrobin
    server fy2 192.168.190.129:80 weight 3 check
    server fy3 192.168.190.130:80 weight 1 check
```

Why: fy2 handles more traffic than fy3, simulating **resource-aware load balancing**.

Step 12 – Restart HAProxy

```
sudo systemctl restart haproxy
```

- Open browser on host:

```
http://<fy1_IP>
```

- Refresh → pages from fy2 and fy3 should alternate.

Why: Confirms load balancing is working correctly.

Step 13 – Verification

1. Check cron jobs:

```
crontab -l
```

2. Check logs on workers:

```
ssh student@192.168.190.129 'cat /tmp/cron_job.log'

ssh student@192.168.190.130 'cat /tmp/cron_job.log'
```

3. Check HAProxy: open browser at http://<fy1_IP> → requests rotate.
   ✅ Demonstrates **distributed task execution and traffic load balancing**.
