We need to create **3 AWS EC2 instances** with the following details:

✅ **Controller** (for application management)
✅ **Slave1** (for monitoring/log scraping)
✅ **Slave2** (for monitoring/log scraping)

Step 1: Create and Launch Instances

1. Open **AWS EC2**.
2. Click **Launch Instance**.
3. **Instance Settings:**
   - **Name:** `controller`
   - **OS Image:** Ubuntu Server 22.04 LTS (HVM), SSD Volume Type (Free Tier)
   - **Instance Type:** `t3.micro` (Free Tier)
   - **Key Pair:** `revatureppk(or create new one)`
   - **Network Settings:** Selected existing security group `ansible-new-sg`
   - **Number of Instances: 3**
4. Click **Launch Instance**.
5. Rename the other two instances:
   - `slave1`
   - `slave2`

Step 2: Connecting to the Controller via PuTTY

1️⃣ Store Private IPs for Reference

- Open **Notepad** and **copy-paste** the private IPs of:
  - **Controller**
  - **Slave1**
  - **Slave2**
  (This helps with SSH setup and hostname configuration.)

2️⃣ Connect to Controller via PuTTY

- Open **AWS EC2 Console** → Copy **Public IP of Controller**.
- Open **PuTTY.exe**:
  - **Host Name (or IP address)** → Paste **Public IP of Controller**.
  - On the left, navigate to:
    - **(+)** `SSH`
    - **(+)** `Auth`
    - **Credentials → Browse for Private Key (`revatureppk.ppk`).`
  - Click **Open**.
  - A **PuTTY Security Alert** pops up → Click **Accept**.

3️⃣ **Log in as Ubuntu User**
In the **PuTTY terminal**, enter:

```
login as: ubuntu
```

Once logged in, set the hostname:

```
sudo hostname controller
```

Switch to the root user:

```
sudo su
```

- *(Prompt should change from `ubuntu@controller` to `root@controller`.)*

📌 Step 3: Configuring `/etc/hosts`

🔹 **Why?**
This ensures the controller can resolve slave machine names by their private IPs.

Open the hosts file:
```
nano /etc/hosts
```

Clear everything inside the file.

Paste the private IPs in this format:
```
<Private IP of Controller>  controller
<Private IP of Slave1>      slave1
<Private IP of Slave2>      slave2
```

Save the file:

Press `Ctrl + O`, then `Enter` (to write changes).

Press `Ctrl + X` (to exit).

📌 Step 4: Configuring SSH (sshd_config)

🔹 **Why?**
To modify SSH settings and allow proper remote access.

Navigate to the SSH directory:
`cd /etc/ssh`

Open the SSH configuration file:

`nano sshd_config`

📌 Step 5: Configuring SSH on Controller, Slave1, and Slave2

🔹 **Why?**
To allow **root login** and **password authentication** for easy SSH communication between the nodes.

1️⃣ Modify SSH Configuration on the Controller

Open the SSH config file:
`nano /etc/ssh/sshd_config`

Make the following changes:

Uncomment `#PermitRootLogin prohibit-password`

Replace `prohibit-password` with `yes`
`PermitRootLogin yes`

Uncomment `#PasswordAuthentication yes`

Comment out `KbdInteractiveAuthentication no`

`#KbdInteractiveAuthentication no`

- **Save and exit:**
  - `Ctrl + O`, `Enter` (to save changes).
  - `Ctrl + X` (to exit).

2️⃣ Modify SSH Client Configuration

Open SSH client config file:

```
nano /etc/ssh/ssh_config
```

Uncomment:
`StrictHostKeyChecking ask`

Change `ask` to `no`
`StrictHostKeyChecking no`

Save and exit:

`Ctrl + O`, `Enter`

`Ctrl + X`

3️⃣ Restart SSH Service

Restart SSH services for changes to take effect:

```
service ssh restart
service sshd restart
```

📌 Step 6: Configure Slave1 & Slave2

🔹 **Why?**
To allow password-based SSH access and ensure name resolution across all nodes.

1️⃣ Connect to Slave1

- **Copy the Public IP of `slave1`** from AWS.
- **Open PuTTY**, enter **Public IP**, and configure:
  - **SSH** → **Auth** → **Credentials** → Browse → Select `revatureppk.ppk`
- Click **Open** and **Accept** security alert.

Log in:
`login as: ubuntu`

Set hostname:
```
sudo hostname slave1
sudo su
```

Modify `/etc/hosts`:
```
nano /etc/hosts
```

Clear everything and paste the **private IP mappings** from Notepad:
```
<Private IP of Controller>  controller
<Private IP of Slave1>      slave1
<Private IP of Slave2>      slave2
```

Save and exit:
`Ctrl + O`, `Enter`, `Ctrl + X`

Modify SSH settings:
```
cd /etc/ssh
nano sshd_config
```

- **Uncomment** `#PermitRootLogin prohibit-password`
- **Replace** `prohibit-password` with `yes`
- **Uncomment** `#PasswordAuthentication yes`
- **Comment out** `KbdInteractiveAuthentication no`

Modify SSH client settings:
```
nano ssh_config
```

- **Uncomment** `StrictHostKeyChecking ask`
- **Change** `ask` to `no`

Restart SSH services:
```
service ssh restart
service sshd restart
```

2️⃣ Repeat the Same Steps for Slave2

- Follow **exactly the same process** for Slave2.

Ensure hostname is set correctly:
```
sudo hostname slave2
```

```
sudo su
```

- Update **SSH configs** and **restart SSH services**.

📌 Step 7: Verify Connectivity

Once SSH is configured on all machines:

On the **Controller**, test connectivity:

```
ping slave1
ping slave2
```

On **Slave1**, test connectivity:

```
ping controller
ping slave2
```

On **Slave2**, test connectivity:

```
ping controller
ping slave1
```

✅ **Success Criteria:**

- All machines should be able to **ping each other** without issues.

📌 Step 8: Set Password for Root User on All Machines

🔹 **Why?**
This ensures that each server has a root password for SSH authentication.

1️⃣ Set a Root Password on Each Machine

On **Controller, Slave1, and Slave2**, run:

```
passwd
```

- **Enter a new password** (twice for confirmation).
- Repeat this step on **all three** machines.

📌 Step 9: Test SSH Access from Controller

🔹 **Why?**
Verify that password-based SSH login works.

On the **Controller**, test SSH access to `Slave1` and `Slave2`:

```
ssh root@slave1
```

- Enter **Slave1's password**.

If successful, exit:
```
exit
```

Repeat for **Slave2**:

```
ssh root@slave2
```

- Enter **Slave2's password**.

If successful, exit:
```
exit
```

✅ **Success Criteria:**

- No errors when logging in with passwords.

📌 Step 10: Generate SSH Key for Key-Based Authentication

🔹 **Why?**
Instead of entering passwords every time, we use an **SSH key pair**.

1️⃣ Generate an SSH Key Pair on Controller

On the **Controller**, run:

```
ssh-keygen
```

When prompted:
`Enter file in which to save the key (/root/.ssh/id_rsa):`

- Just **press Enter** (default location).
- No need to enter a passphrase—just **press Enter** twice.

✅ **This generates:**

- A **private key** (`/root/.ssh/id_rsa`).
- A **public key** (`/root/.ssh/id_rsa.pub`).

📌 Step 11: Copy SSH Key to All Machines

🔹 **Why?**
Allows **passwordless SSH login** from Controller to all other machines.

1️⃣ Navigate to SSH Directory

```
cd /root/.ssh/
```

2️⃣ Copy SSH Key to Controller

```
ssh-copy-id root@controller
```

- Enter **Controller's root password**.

3️⃣ Copy SSH Key to Slave1

```
ssh-copy-id root@slave1
```

- Enter **Slave1's root password**.

4️⃣ Copy SSH Key to Slave2

```
ssh-copy-id root@slave2
```

- Enter **Slave2's root password**.

✅ **Now the Controller can SSH into Slave1 and Slave2 without a password.**

📌 Step 12: Verify Key-Based SSH Login

🔹 **Why?**
To confirm that SSH key authentication is working.

1️⃣ Change Directory

```
cd /home/ubuntu
```

2️⃣ SSH into Slave1 Without a Password

```
ssh root@slave1
```

- **You should log in without entering a password**.

If successful, type:
```
exit
```

3️⃣ SSH into Slave2 Without a Password

```
ssh root@slave2
```

- **You should log in without entering a password**.

If successful, type:
```
exit
```

✅ **Success Criteria:**

- No password prompts when connecting to `slave1` and `slave2` from the **Controller**.

Step 13: Install Ansible on the Controller

🔹 **Why?**
The **Controller** will use Ansible to **automate deployments** across all machines.

1️⃣ Update Package Lists & Install Required Packages

Run the following **on the Controller**:

```
sudo apt update
sudo apt install software-properties-common
sudo add-apt-repository --yes --update ppa:ansible/ansible
sudo apt install ansible
```

✅ **Ansible is now installed on the Controller**.

📌 Step 14: Configure Ansible Inventory

🔹 **Why?**
The **Ansible inventory** file (`hosts`) defines groups of servers to automate.

1️⃣ Open the Inventory File

```
nano hosts
```

2️⃣ Add the Following Configuration

```
[all]
controller
slave1
slave2

[con]
controller

[slaves]
slave1
slave2
```

🔹 **Explanation:**

- **[all]** → Lists all servers.
- **[con]** → Defines the **Controller group**.
- **[slaves]** → Defines the **Slave group**.

3️⃣ Save and Exit

- `Ctrl + O`, then `Enter` (save)
- `Ctrl + X` (exit)

📌 Step 15: Test Ansible Connectivity

🔹 **Why?**
To verify that **Ansible can communicate** with all nodes.

Run:

```
ansible -i hosts all -m ping
```

✅ **Success Criteria:**

- If everything is working, you should see a **green "pong" response** from all servers.

📌 Step 16: Install MySQL and Python Dependencies

🔹 **Why?**
We need **MySQL for log storage** and **Python dependencies** for log scraping.

1️⃣ Update Package Lists

```
apt update
```

2️⃣ Install MySQL and Python Dependencies

```
apt install -y mysql-server python3-pip
```

- When prompted, **press Tab** and select **OK**.

3️⃣ Check MySQL Service Status

```
systemctl status mysql
```

✅ **Ensure MySQL is running**.

📌 Step 17: Configure MySQL Database

🔹 **Why?**
Create a database and a **user for metrics collection**.

1️⃣ Log into MySQL

```
mysql -u root -p
```

- **Enter the root password**.

2️⃣ Create the Database

```
CREATE DATABASE system_metrics;
```

3️⃣ Create a User

```
CREATE USER 'metrics_user'@'%' IDENTIFIED BY '1234';
```

4️⃣ Grant Full Privileges to the User

```
GRANT ALL PRIVILEGES ON system_metrics.* TO 'metrics_user'@'%' WITH
GRANT OPTION;
```

5️⃣ Apply Changes

```
FLUSH PRIVILEGES;
EXIT;
```

✅ **MySQL is now set up with a database and user.**

📌 Step 18: Modify MySQL Configuration for Remote Access

🔹 **Why?**
By default, MySQL only allows local connections (`127.0.0.1`).
We change it to `0.0.0.0` to allow remote access from **slaves**.

1️⃣ Navigate to the MySQL Config Directory

```
cd /etc/mysql/
cd mysql.conf.d/
```

2️⃣ Edit the MySQL Configuration File

```
nano mysqld.cnf
```

Find the line:
`bind-address = 127.0.0.1`
Change it to:

```
bind-address = 0.0.0.0
```

Save & Exit:

`Ctrl + O`, `Enter` (save)

`Ctrl + X` (exit)

3️⃣ Restart MySQL

```
systemctl restart mysql
```

✅ **Now MySQL can accept remote connections.**

📌 Step 19: Configure Ansible Playbook for Metrics Collection

🔹 **Why?**
We use **Ansible** to **automate the deployment of a Python script** that collects system metrics.

1️⃣ Go to the Home Directory

```
cd /home/ubuntu
```

2️⃣ Create the Ansible Playbook

```
nano metrics.yaml
```

3️⃣ Copy & Paste the Following Configuration

```
---
  - name: Setup Linux Metrics Collection on Multiple Servers
    hosts: slaves
    become: yes
    tasks:

      - name: Update Workers
        apt:
          update_cache: yes
          upgrade: 'yes'

      - name: Install required packages
        apt:
          name:
            - python3
            - python3-pip
          state: present

      - name: Install Python Dependencies
        pip:
          name:
            - psutil
            - mysql-connector-python

      - name: Deploy Python Script for Metrics Collection
        copy:
          dest: /opt/linux_metrics.py
          mode: "0755"
          content: |
```

```
import psutil
import mysql.connector
from datetime import datetime
import socket

# Database Configuration (Central MySQL Server)
db_config = {
  "host": "ANSIBLE CONTROLLER IP",
  "user": "metrics_user",
  "password": "MYSQL USER PASSWORD",
  "database": "system_metrics"
}

def get_system_metrics():
    cpu_usage = psutil.cpu_percent(interval=1)
    memory_usage = psutil.virtual_memory().percent
    disk_usage = psutil.disk_usage('/').percent
    hostname = socket.gethostname()
    return hostname, cpu_usage, memory_usage, disk_usage

def insert_into_db(hostname, cpu, memory, disk):
    try:
        conn = mysql.connector.connect(**db_config)
        cursor = conn.cursor()
        sql = "INSERT INTO metrics (server_name,
cpu_usage, memory_usage, disk_usage) VALUES (%s, %s, %s, %s)"
        cursor.execute(sql, (hostname, cpu, memory, disk))
        conn.commit()
        cursor.close()
        conn.close()
        print(f"[{datetime.now()}] {hostname} -
CPU={cpu}%, RAM={memory}%, Disk={disk}%")
    except Exception as e:
        print("Database Error:", e)

if __name__ == "__main__":
    hostname, cpu, memory, disk = get_system_metrics()
    insert_into_db(hostname, cpu, memory, disk)
```

```
  - name: Setup Cron Job for Data Collection
    cron:
      name: "Collect and Send Linux Metrics"
      minute: "*/1"
      job: "/usr/bin/python3 /opt/linux_metrics.py"
```

4️⃣ Save & Exit

- `Ctrl + O`, `Enter` (save)
- `Ctrl + X` (exit)

✅ **Now the Ansible playbook is ready!**

📌 Step 20: Deploy the Ansible Playbook

🔹 **Why?**
To execute the `metrics.yaml` playbook and install necessary packages on **Slave1 & Slave2**.

1️⃣ Run the Ansible Playbook in Dry Run Mode

```
ansible-playbook -i hosts metrics.yaml --check
```

✅ **This checks for potential issues before making real changes.**

2️⃣ Run the Playbook for Real Deployment

```
ansible-playbook -i hosts metrics.yaml
```

⚠️ **Issue:** The package installation **failed** on slaves.

📌 Step 21: Fix the Package Installation Issue

🔹 **Why?**
The slaves were missing an `apt update`, causing package installation failures.

1️⃣ Manually Update Packages on Slave1

```
ssh root@slave1
cd /home/ubuntu
apt update
exit
```

2️⃣ Manually Update Packages on Slave2

```
ssh root@slave2
cd /home/ubuntu
apt update
exit
```

✅ **Now, both slaves have updated package lists.**

📌 Step 22: Re-run the Ansible Playbook

🔹 **Why?**
Now that package updates are fixed, we retry the deployment.

```
ansible-playbook -i hosts metrics.yaml
```

✅ **This time, the playbook should complete successfully.**

📌 Step 23: Verify MySQL Table Setup

🔹 **Why?**
Create a table to store **system metrics**.

1️⃣ Log in to MySQL

```
mysql -u metrics_user -p
```

- Enter **password: 1234** (or whatever was set earlier).

2️⃣ Show Available Databases

```
SHOW DATABASES;
```

3️⃣ Create the `metrics` Table

```
CREATE TABLE IF NOT EXISTS metrics (
    id INT AUTO_INCREMENT PRIMARY KEY,
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
    server_name VARCHAR(255),
    cpu_usage FLOAT,
    memory_usage FLOAT,
    disk_usage FLOAT
);
```

⚠️ **ERROR 1046 (3D000): No database selected**

📌 Step 24: Select the Correct Database

1️⃣ Switch to the `system_metrics` Database

```
USE system_metrics;
```

2️⃣ Re-run the `CREATE TABLE` Command

```
CREATE TABLE IF NOT EXISTS metrics (
    id INT AUTO_INCREMENT PRIMARY KEY,
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
    server_name VARCHAR(255),
    cpu_usage FLOAT,
    memory_usage FLOAT,
    disk_usage FLOAT
);
```

✅ **Table should now be created successfully.**

📌 Step 25: Verify Data Collection

🔹 **Why?**
Check if **system metrics are being stored** in the database.

1️⃣ Check for Stored Metrics

```
SELECT * FROM metrics;
```

2️⃣ Order Results by Latest Entry

```
SELECT * FROM metrics ORDER BY timestamp DESC;
```

✅ **If everything is working, you should see rows of collected metrics.**

📌 Step 26: Wait for Metrics Collection & Re-Check

🔹 **Why?**
Since the cron job **collects data every minute**, we need to wait.

1️⃣ **Run the Playbook Again**

```
ansible-playbook -i hosts metrics.yaml
```

2️⃣ **Wait for 1 Minute**
(The cron job runs the Python script every minute to insert new data.)

3️⃣ **Log in to MySQL Again**

```
mysql -u metrics_user -p
USE system_metrics;
```

4️⃣ **Check the Latest Entries**

```
SELECT * FROM metrics ORDER BY timestamp DESC;
```

✅ **Success Criteria:**

- If new rows are appearing **every minute**, the data collection system is working! 🎯

📌 Step 27: Download and Extract Prometheus

🔹 **Why?**
Prometheus is used for **real-time system monitoring and alerting**.

1️⃣ Download Prometheus

- Go to [Prometheus Downloads](#).
- Find the latest version (e.g., `prometheus-2.53.3.linux-amd64.tar.gz`).
- Copy the **download link**.

2️⃣ Download Using `wget`

```
wget
https://github.com/prometheus/prometheus/releases/download/v2.53.3/pro
metheus-2.53.3.linux-amd64.tar.gz
```

✅ **This downloads Prometheus to the current directory.**

3️⃣ Extract the Archive

```
tar xvfz prometheus-2.53.3.linux-amd64.tar.gz
```

✅ **This extracts Prometheus files into a folder.**

4️⃣ Remove the Original Archive

```
rm prometheus-2.53.3.linux-amd64.tar.gz
```

✅ **This cleans up the directory.**

5️⃣ Rename the Extracted Folder

```
mv prometheus-2.53.3.linux-amd64/ prometheus
```

✅ **This renames the extracted folder to just `prometheus` for easier access.**

📌 Step 28: Create a Prometheus System User

🔹 **Why?**
To run Prometheus **securely** without root privileges.

```
sudo useradd --no-create-home --shell /bin/false prometheus
```

✅ **This creates a system user named `prometheus`.**

📌 Step 29: Set Up Required Directories

🔹 **Why?**
Prometheus needs configuration and storage directories.

```
sudo mkdir /etc/prometheus
sudo mkdir /var/lib/prometheus
```

✅ **Creates directories for Prometheus configuration and data storage.**

📌 Step 30: Assign Permissions to Prometheus User

```
sudo chown prometheus:prometheus /etc/prometheus
sudo chown prometheus:prometheus /var/lib/prometheus
```

✅ **Grants the `prometheus` user ownership of its directories.**

📌 Step 31: Move Prometheus Binaries

🔹 **Why?**
To make Prometheus **accessible system-wide**.

```
sudo cp prometheus/prometheus /usr/local/bin/
sudo cp prometheus/promtool /usr/local/bin/
```

✅ **Copies Prometheus executables to `/usr/local/bin/`.**

1️⃣ Change Ownership of Binaries

```
sudo chown prometheus:prometheus /usr/local/bin/prometheus
sudo chown prometheus:prometheus /usr/local/bin/promtool
```

✅ **Ensures the `prometheus` user owns these files.**

📌 Step 32: Copy Console Files for Prometheus Web UI

🔹 **Why?**
These files are needed for **Prometheus' built-in UI**.

```
sudo cp -r prometheus/consoles /etc/prometheus
sudo cp -r prometheus/console_libraries /etc/prometheus
```

✅ **Copies console templates and libraries to `/etc/prometheus/`.**

1️⃣ Set Ownership for Console Files

```
sudo chown -R prometheus:prometheus /etc/prometheus/consoles
sudo chown -R prometheus:prometheus /etc/prometheus/console_libraries
```

✅ **Ensures the `prometheus` user owns these directories.**

📌 Step 33: Create a Prometheus Systemd Service

🔹 **Why?**
To manage Prometheus **as a system service**.

1️⃣ Open Systemd Service File

```
sudo nano /etc/systemd/system/prometheus.service
```

2️⃣ Add the Following Configuration

```
[Unit]
Description=Prometheus
Wants=network-online.target
After=network-online.target

[Service]
User=prometheus
Group=prometheus
Type=simple
ExecStart=/usr/local/bin/prometheus \
    --config.file /etc/prometheus/prometheus.yml \
    --storage.tsdb.path /var/lib/prometheus/ \
    --web.console.templates=/etc/prometheus/consoles \
    --web.console.libraries=/etc/prometheus/console_libraries

[Install]
WantedBy=multi-user.target
```

3️⃣ Save & Exit

- `Ctrl + O`, `Enter` (save).
- `Ctrl + X` (exit).

✅ **This defines how Prometheus will run as a service.**

📌 Step 34: Reload and Start Prometheus Service

🔹 **Why?**
To apply the systemd configuration and start Prometheus properly.

1️⃣ Reload Systemd Daemon

```
sudo systemctl daemon-reload
```

✅ **Reloads systemd to recognize new service configurations.**

2️⃣ Start Prometheus

```
sudo systemctl start prometheus
```

❌ **This gave an error.**

📌 Step 35: Fixing the Prometheus Installation

🔹 **Why?**
Prometheus files might be missing from `/etc/prometheus/`.

1️⃣ Navigate to the Prometheus Directory

```
cd prometheus/
```

2️⃣ Copy All Prometheus Files to `/etc/prometheus/`

```
cp -r * /etc/prometheus/
```

✅ **Ensures all required files are available for Prometheus to run.**

3️⃣ Reload Systemd Again

```
sudo systemctl daemon-reload
```

4️⃣ Check the Status of Prometheus

```
systemctl status prometheus
```

❌ **Failed initially.**

5️⃣ Restart Prometheus

```
systemctl restart prometheus
```

6️⃣ Verify Prometheus is Running

```
systemctl status prometheus
```

✅ **Now it is active and running!**

📌 Step 36: Access Prometheus Web UI

🔹 **Why?**
To verify Prometheus is working.

1️⃣ Get Public IP of Controller

- Go to **AWS EC2 Console**.
- Copy the **Public IP Address** of `controller`.

2️⃣ Open Prometheus in Browser

In the browser, enter:
`http://<controller-public-ip>:9090`

-

✅ **Prometheus UI should load successfully!** 🎯

📌 Step 37: Deploy Node Exporter on Slaves

🔹 **Why?**
Node Exporter collects **CPU, memory, disk, and network usage** metrics.

1️⃣ Navigate Back to Home Directory

```
cd ..
```

2️⃣ Remove Old Prometheus Directory

```
rm -rf prometheus/
```

✅ **Cleans up unnecessary files.**

📌 Step 38: Create Ansible Playbook for Node Exporter

🔹 **Why?**
To **automate** the deployment of Node Exporter.

1️⃣ Create the Node Exporter Ansible Playbook

```
nano node.yaml
```

2️⃣ Copy and Paste the Following Configuration

```
---
- name: Install and Configure Prometheus Node Exporter on Ubuntu
  hosts: slaves
  become: yes
  tasks:
    - name: Update package lists
      apt:
        update_cache: yes

    - name: Install required dependencies
      apt:
        name:
          - wget
          - tar
        state: present
```

```
    - name: Download Node Exporter
      get_url:
        url:
"https://github.com/prometheus/node_exporter/releases/download/v1.7.0/
node_exporter-1.7.0.linux-amd64.tar.gz"
        dest: "/tmp/node_exporter.tar.gz"

    - name: Extract Node Exporter
      ansible.builtin.unarchive:
        src: "/tmp/node_exporter.tar.gz"
        dest: "/usr/local/bin/"
        remote_src: yes
        extra_opts: [--strip-components=1]

    - name: Create Node Exporter system user
      user:
        name: node_exporter
        system: yes
        shell: /bin/false

    - name: Set permissions for Node Exporter binary
      file:
        path: "/usr/local/bin/node_exporter"
        owner: node_exporter
        group: node_exporter
        mode: "0755"

    - name: Create systemd service file for Node Exporter
      copy:
        dest: "/etc/systemd/system/node_exporter.service"
        mode: "0644"
        content: |
          [Unit]
          Description=Prometheus Node Exporter
          Wants=network-online.target
          After=network-online.target
```

```
        [Service]
        User=node_exporter
        Group=node_exporter
        Type=simple
        ExecStart=/usr/local/bin/node_exporter

        [Install]
        WantedBy=multi-user.target

    - name: Reload systemd daemon
      systemd:
        daemon_reload: yes

    - name: Enable and start Node Exporter service
      systemd:
        name: node_exporter
        state: started
        enabled: yes
```

3️⃣ Save & Exit

- `Ctrl + O`, `Enter` (save).
- `Ctrl + X` (exit).

✅ **Now the Ansible playbook is ready!**

📌 Step 39: Deploy Node Exporter Using Ansible

1️⃣ Run the Playbook in Dry Run Mode

```
ansible-playbook -i hosts node.yaml --check
```

✅ **Checks for potential issues before execution.**

2️⃣ Run the Playbook for Real Deployment

```
ansible-playbook -i hosts node.yaml
```

✅ **Installs and starts Node Exporter on Slave1 and Slave2.**

📌 Step 40: Verify Node Exporter on Slave1

🔹 **Why?**
To confirm that **Node Exporter is running** on Slave1.

1️⃣ Check Node Exporter Status on Slave1

```
systemctl status node_exporter
```

✅ **If active, Node Exporter is working properly.**

📌 Step 41: Configure Prometheus to Scrape Node Exporter

🔹 **Why?**
To **collect metrics** from Slave1 and Slave2 in Prometheus.

1️⃣ Go to Prometheus Configuration Directory

```
cd /etc/prometheus/
```

2️⃣ Open `prometheus.yml` for Editing

```
nano prometheus.yml
```

3️⃣ Add the Following at the End

```
- job_name: "node_exporter"
  # metrics_path defaults to 'metrics'
  # scheme defaults to 'http'.
  static_configs:
```

```
    - targets: ["<PUBLIC_IP_OF_SLAVE1>:9100"]

- job_name: "node_exporter2"
  # metrics_path defaults to 'metrics'
  # scheme defaults to 'http'.
  static_configs:
    - targets: ["<PUBLIC_IP_OF_SLAVE2>:9100"]
```

🔹 **Replace** `<PUBLIC_IP_OF_SLAVE1>` and `<PUBLIC_IP_OF_SLAVE2>` with the **actual public IPs** from AWS.

4️⃣ Save & Exit

- `Ctrl + O`, `Enter` (save).
- `Ctrl + X` (exit).

📌 Step 42: Restart Prometheus

🔹 **Why?**
To apply the new configuration.

```
systemctl restart prometheus
```

✅ **Prometheus should now scrape data from Node Exporter running on both slaves.**

📌 Step 43: Verify Node Exporter in Prometheus UI

1️⃣ Open Prometheus Web UI

Go to your browser and enter:
`http://<CONTROLLER_PUBLIC_IP>:9090`

2️⃣ Check the Node Exporter Targets

- Click **Status** → **Targets**.

- **Expected Result:** Both `node_exporter` and `node_exporter2` should show **"UP"**. ✅

📌 Step 44: Install Prometheus Alertmanager

🔹 **Why?**
Alertmanager **sends notifications** when metrics exceed predefined thresholds.

1️⃣ Download Alertmanager

- Go to [Prometheus Downloads](#).
- Find **Alertmanager** (e.g., `alertmanager-0.28.0.linux-amd64.tar.gz`).
- Copy the **download link**.

2️⃣ Switch to Home Directory

```
cd /home/ubuntu
```

3️⃣ Download Alertmanager

```
wget https://github.com/prometheus/alertmanager/releases/download/v0.28.0/alertmanager-0.28.0.linux-amd64.tar.gz
```

✅ **Downloads Alertmanager to the home directory.**

📌 Step 45: Extract and Organize Alertmanager

1️⃣ Extract Alertmanager

```
tar xvfz alertmanager-0.28.0.linux-amd64.tar.gz
```

✅ **Extracts the files into a new folder.**

2️⃣ Remove the Original Archive

```
rm alertmanager-0.28.0.linux-amd64.tar.gz
```

✅ **Cleans up unnecessary files.**

3️⃣ Rename the Extracted Folder

```
mv alertmanager-0.28.0.linux-amd64/ alertmanager
```

✅ **Renames the directory to `alertmanager` for easier access.**

📌 Step 46: Move Alertmanager Binaries to System Path

🔹 **Why?**
To make Alertmanager **accessible system-wide**.

1️⃣ Copy Alertmanager Binaries to `/usr/bin/`

```
sudo cp alertmanager/alertmanager /usr/bin/
sudo cp alertmanager/amtool /usr/bin/
```

✅ **Now Alertmanager and its command-line tool (`amtool`) can be accessed from anywhere.**

sudo useradd --no-create-home --shell /bin/false alertmanager

---

📌 Step 47: Assign Correct Ownership to Alertmanager

🔹 **Why?**
Ensures Alertmanager runs under its own system user.

```
sudo chown alertmanager:alertmanager /usr/bin/alertmanager
sudo chown alertmanager:alertmanager /usr/bin/amtool
```

✅ **Now, only the `alertmanager` user can execute these files.**

📌 Step 48: Move and Assign Alertmanager Configuration File

🔹 **Why?**
Alertmanager needs a **config file (`alertmanager.yml`)** to define alert rules.

1️⃣ Copy Configuration File

```
sudo cp alertmanager-files/alertmanager.yml
/etc/alertmanager/alertmanager.yml
```

2️⃣ Set Correct Permissions

```
sudo chown alertmanager:alertmanager
/etc/alertmanager/alertmanager.yml
```

✅ **Now Alertmanager can read its configuration file.**

📌 Step 49: Create a Systemd Service for Alertmanager

🔹 **Why?**
To run Alertmanager as a **background service**.

1️⃣ Create a New Systemd Service File

```
sudo nano /etc/systemd/system/alertmanager.service
```

2️⃣ Paste the Following Configuration

```
[Unit]
Description=AlertManager
Wants=network-online.target
After=network-online.target

[Service]
User=alertmanager
Group=alertmanager
Type=simple
```

```
ExecStart=/usr/bin/alertmanager \
    --config.file /etc/alertmanager/alertmanager.yml \
    --storage.path /var/lib/alertmanager/

[Install]
WantedBy=multi-user.target
```

3️⃣ Save & Exit

- `Ctrl + O`, `Enter` (save).
- `Ctrl + X` (exit).

✅ **This ensures Alertmanager starts automatically on system boot.**

📌 Step 50: Start Alertmanager

1️⃣ Reload Systemd Daemon

```
sudo systemctl daemon-reload
```

2️⃣ Start Alertmanager

```
sudo systemctl start alertmanager
```

3️⃣ Verify Alertmanager Status

```
systemctl status alertmanager
```

✅ **If running, Alertmanager is successfully installed!** 🎯

📌 Step 51: Configure Alert Rules in Prometheus

🔹 **Why?**
To **trigger alerts** when Node Exporter **goes down**.

1️⃣ Go to Prometheus Configuration Directory

```
cd /etc/prometheus/
```

2️⃣ Create a Rules Directory

```
mkdir rules
```

3️⃣ Navigate to the Rules Directory

```
cd rules
```

4️⃣ Create a New Rules File

```
nano rules.yaml
```

5️⃣ Paste the Following Alerting Rules

```
groups:
  - name: my_rule
    rules:
      - alert: Node_exporter_down
        expr: up{job="node_exporter"} == 0
      - alert: Node_exporter2_down
        expr: up{job="node_exporter_2"} == 0
```

6️⃣ Save & Exit

- `Ctrl + O`, `Enter` (save).
- `Ctrl + X` (exit).

✅ **These rules trigger an alert when either `node_exporter` or `node_exporter_2` goes down.**

📌 Step 52: Connect Prometheus to Alertmanager

🔹 **Why?**
To **send alerts from Prometheus** to Alertmanager.

1️⃣ Navigate Back to Prometheus Directory

```
cd ..
```

2️⃣ Edit Prometheus Configuration

```
nano prometheus.yml
```

3️⃣ Find the Alertmanager Configuration Section

```
# Alertmanager configuration
```

4️⃣ Modify the Target to Use the Controller's Public IP

```
# - alertmanager:9093
- targets: ["<PUBLIC_IP_OF_CONTROLLER>:9093"]
```

- 🔹 **Replace** `<PUBLIC_IP_OF_CONTROLLER>` with the **actual public IP** from AWS.

5️⃣ Save & Exit

- `Ctrl + O`, `Enter` (save).
- `Ctrl + X` (exit).

📌 Step 53: Validate Alerting Rules in Prometheus

🔹 **Why?**
To ensure the alerting rules **are correctly formatted**.

1️⃣ Navigate to Rules Directory

```
cd rules/
```

2️⃣ Edit `rules.yaml` to Confirm Alert Rules

```
nano rules.yaml
```

3️⃣ Paste or Confirm the Alerting Rules

```
groups:
  - name: my_rule
    rules:
      - alert: Node_exporter_down
        expr: up{job="node_exporter"} == 0
      - alert: Node_exporter_2_down
        expr: up{job="node_exporter_2"} == 0
```

4️⃣ Save & Exit

- `Ctrl + O`, `Enter` (save).
- `Ctrl + X` (exit).

5️⃣ Go Back to Prometheus Directory

```
cd ..
```

6️⃣ Validate the Alert Rules Using `promtool`

```
./promtool check rules/rules.yaml
```

✅ **If no errors are displayed, the rules are correctly formatted.**

📌 Step 54: Update Prometheus to Use the New Rules

🔹 **Why?**
To enable Prometheus to load the custom alert rules.

1️⃣ Edit `prometheus.yml`

```
nano prometheus.yml
```

2️⃣ Locate and Modify the Rules Section

Find:

```
# - "first_rules.yml"
```

Replace it with:

```
- "rules/rules.yaml"
```

3️⃣ Save & Exit

- `Ctrl + O`, `Enter` (save).
- `Ctrl + X` (exit).

4️⃣ Restart Prometheus and Alertmanager

```
systemctl restart prometheus
systemctl restart alertmanager
```

✅ **Prometheus and Alertmanager are now using the new alert rules.**

📌 Step 55: Verify Alerts in Prometheus UI

1️⃣ Open Prometheus in Browser

Enter:
`http://<CONTROLLER_PUBLIC_IP>:9090`

- Click **Alerts**.
- You should see:
  - `node_exporter_down`
  - `node_exporter_2_down`

✅ **This confirms that Prometheus is now detecting alerts.**

📌 Step 56: Verify Alertmanager UI

1️⃣ Open Alertmanager in Browser

Enter:
`http://<CONTROLLER_PUBLIC_IP>:9093`

✅ **Alertmanager should now be running.**

📌 Step 57: Modify Alert Rules to Add a Delay

🔹 **Why?**
To trigger alerts **only if Node Exporter remains down for 1 minute**.

1️⃣ Navigate to Rules Directory

```
cd rules
```

2️⃣ Edit `rules.yaml`

```
nano rules.yaml
```

3️⃣ Modify Alert Rules to Add a `for` Clause

```
groups:
  - name: my_rule
    rules:
      - alert: Node_exporter_down
        expr: up{job="node_exporter"} == 0
        for: 1m

      - alert: Node_exporter2_down
        expr: up{job="node_exporter_2"} == 0
        for: 1m
```

4️⃣ Save & Exit

- `Ctrl + O`, `Enter` (save).
- `Ctrl + X` (exit).

5️⃣ Restart Prometheus & Alertmanager

```
cd ..
systemctl restart prometheus
systemctl restart alertmanager
```

✅ **Now, alerts will only trigger if Node Exporter is down for 1 minute.**

📌 Step 58: Configure Alertmanager for Email Notifications

🔹 **Why?**
To **send email alerts** when Prometheus detects a problem.

1️⃣ Navigate to Alertmanager Directory

```
cd alertmanager/
```

2️⃣ Edit Alertmanager Configuration

```
nano alertmanager.yml
```

3️⃣ Remove Everything & Paste the Following

```
route:
    receiver: admin

receivers:
- name: admin
  email_configs:
  - to: "email@gmail.com"
    from: "youremail@gmail.com"
    smarthost: smtp.gmail.com:587
    auth_username: "youremail@gmail.com"
    auth_identity: "youremail@gmail.com"
    auth_password: "app_password"
```

🔹 **Replace:**

- `"youremail@gmail.com"` with **your actual Gmail address**.
- `"app_password"` with **your generated Google App Password**.

4️⃣ Save & Exit

- `Ctrl + O`, `Enter` (save).
- `Ctrl + X` (exit).

📌 Step 59: Generate a Gmail App Password

🔹 **Why?**
Google **blocks less secure apps** from sending emails with your normal password. You need to create an **App Password**.

1️⃣ Create a Google App Password

- Go to **Google App Passwords**.
- Sign in to your **Google account**.
- Under **Select App**, choose **Other (Custom Name)**.
- Enter **"Prometheus"** and click **Generate**.
- **Copy the generated password** and paste it in `auth_password`.

✅ **Now, Alertmanager can send emails via Gmail.**

📌 Step 60: Restart Alertmanager

```
systemctl restart alertmanager
```

✅ **Now, if an alert is triggered, you will receive an email notification!** 🎯

📌 Step 61: Install Grafana on the Controller

🔹 **Why?**
Grafana provides **real-time visualization** for Prometheus metrics.

1️⃣ Navigate to Home Directory

```
cd /home/ubuntu
```

2️⃣ Update Package Lists

```
apt update
```

3️⃣ Install Required Dependencies

```
apt-get install -y apt-transport-https
apt-get install -y software-properties-common wget
```

4️⃣ Add Grafana GPG Key

```
wget -q -O - https://packages.grafana.com/gpg.key | apt-key add -
```

✅ **This ensures that Grafana packages are trusted.**

5️⃣ Add Grafana to the Package Repository

```
echo "deb https://packages.grafana.com/oss/deb stable main" | tee -a
/etc/apt/sources.list.d/grafana.list
```

6️⃣ Update Packages Again

```
apt update
```

7️⃣ Install Grafana

```
apt install grafana -y
```

8️⃣ Enable and Start Grafana Service

```
systemctl enable grafana-server
systemctl start grafana-server
```

✅ **Grafana is now running!** 🎯

Navigate to Prometheus Directory

```
cd /etc/prometheus/
```

5️⃣ Restart Alertmanager

```
systemctl restart alertmanager
```

📌 Step 62: Start Node Exporter on Slave1

🔹 **Why?**
Ensures that **Node Exporter is actively collecting system metrics**.

1️⃣ Start Node Exporter

```
start node_exporter
```

✅ **Now Prometheus can collect metrics from this instance.**

📌 Step 63: Restart Prometheus & Alertmanager

🔹 **Why?**
To apply changes and ensure alerts and metrics are up-to-date.

```
systemctl restart alertmanager
systemctl restart prometheus
```

✅ **Everything is now restarted and running properly.**

📌 Step 64: Access Grafana Web UI

🔹 **Why?**
To set up **Prometheus as a data source** in Grafana.

1️⃣ Open Grafana in a Browser

Go to:
`http://<CONTROLLER_PUBLIC_IP>:3000`

- **Skip password setup**.

📌 Step 65: Connect Grafana to Prometheus

🔹 **Why?**
To visualize system metrics from Prometheus.

1️⃣ Add a New Data Source

- Click **Connections** → **Add New Connection**.
- Select **Prometheus**.
- Click **Add New Data Source**.

2️⃣ Configure Prometheus URL

In **Connection** → **Prometheus Server URL**, enter:
`http://<CONTROLLER_PUBLIC_IP>:9090`

-
- Click **Save & Test**.

✅ **Now Grafana is connected to Prometheus!** 🎯

📌 Step 66: Explore & Query System Metrics

🔹 **Why?**
To verify that **Prometheus is collecting metrics correctly**.

1️⃣ Go to Explore

- Click **Explore**.
- In the **Outline Section**, select **Prometheus**.
- Change **Refresh Interval** (far right) to **5s**.

2️⃣ Query a Metric

- Click **Metrics** → **Let's Start**.

- In **Data Source**, select **Prometheus**.

Search for:
`node_memory_Active_anon_bytes`

- Change time range to **Last 5 Minutes**.

✅ **Now you can see system memory usage in real-time!** 🔥

📌 Step 67: Create a Dashboard for Node Exporter Metrics

🔹 **Why?**
To monitor **Active Memory usage** of **both Node Exporters**.

1️⃣ Go to Explore

Select **Metric**:
`node_memory_Active_bytes`
In **Label Filters**, set:

```
instance = <SLAVE1_PUBLIC_IP>:9100
```

Add a second query:
`node_memory_Active_bytes`

In **Label Filters**, set:
`instance = <SLAVE2_PUBLIC_IP>:9100`

Click **Add** → **Add to Dashboard**.

2️⃣ Open & Customize Dashboard

- Click **Open Dashboard**.
- Click **Edit**.
- Change visualization type from **Table** to **Time Series**.
- **Deselect** "Table View".

Change the **Title** to:
`Active Memory of Node Exporter 1 and 2`

3️⃣ Save Dashboard

- Click **Save Dashboard** → **Save**.

✅ **Now, your Grafana dashboard is displaying real-time system memory usage for both Node Exporters!** 🎯