

# Version Control and Management System

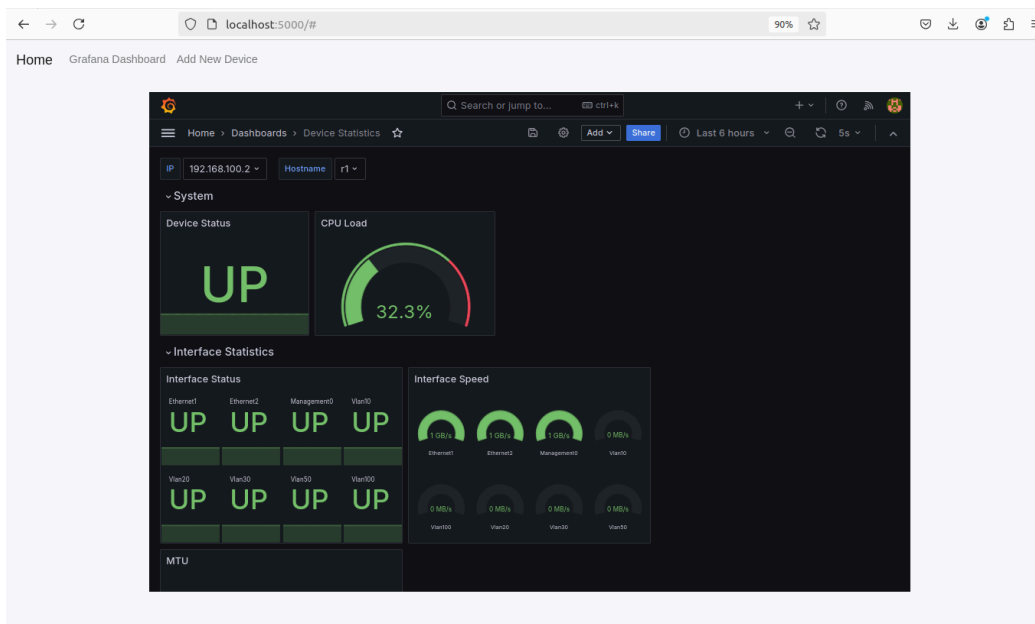
I am utilizing my personal GitHub repository to manage and store all configuration files and automation scripts. You can access the repository here: [GitHub Repository](#).

## Automation Framework

### Grafana Setup

I used **Grafana** to visualize network metrics collected from various network devices. By integrating Grafana with the backend, I can monitor key metrics like **interface statuses**, **CPU utilization**, **packets transmitted**, and more, all in real-time.

- The **Grafana** dashboard is embedded in the web interface. I've customized the navigation bar to include a "Grafana Dashboard" tab, which, when clicked, loads the Grafana dashboard in an iframe.
- The dashboard pulls data from **Prometheus** and **InfluxDB**, which collects metrics from devices using **SNMP** and **gNMI** protocols.



Screenshot 1: Grafana embedded in the website

## Metrics Tracked

- **MTU** values for all interfaces
- **Interface status** (*Up/Down*)
- **Packets in/out** per interface
- **Packet discards** per interface
- **CPU utilization** across devices
- **System uptime**

## Backend and Frontend Integration

The **backend** is built using Flask, while the **frontend** uses HTML, CSS, and JavaScript. The goal was to provide a user-friendly interface to manage network devices, add new configurations, and visualize the system.

## Frontend Overview

The **Add New Device** form allows users to add network devices of different types: **Access**, **Core**, and **Edge** routers.

Each router type has a dedicated form, with **dynamic form sections** (e.g., *VLANs*, *interfaces*, *static routes*) that appear or hide based on user input.

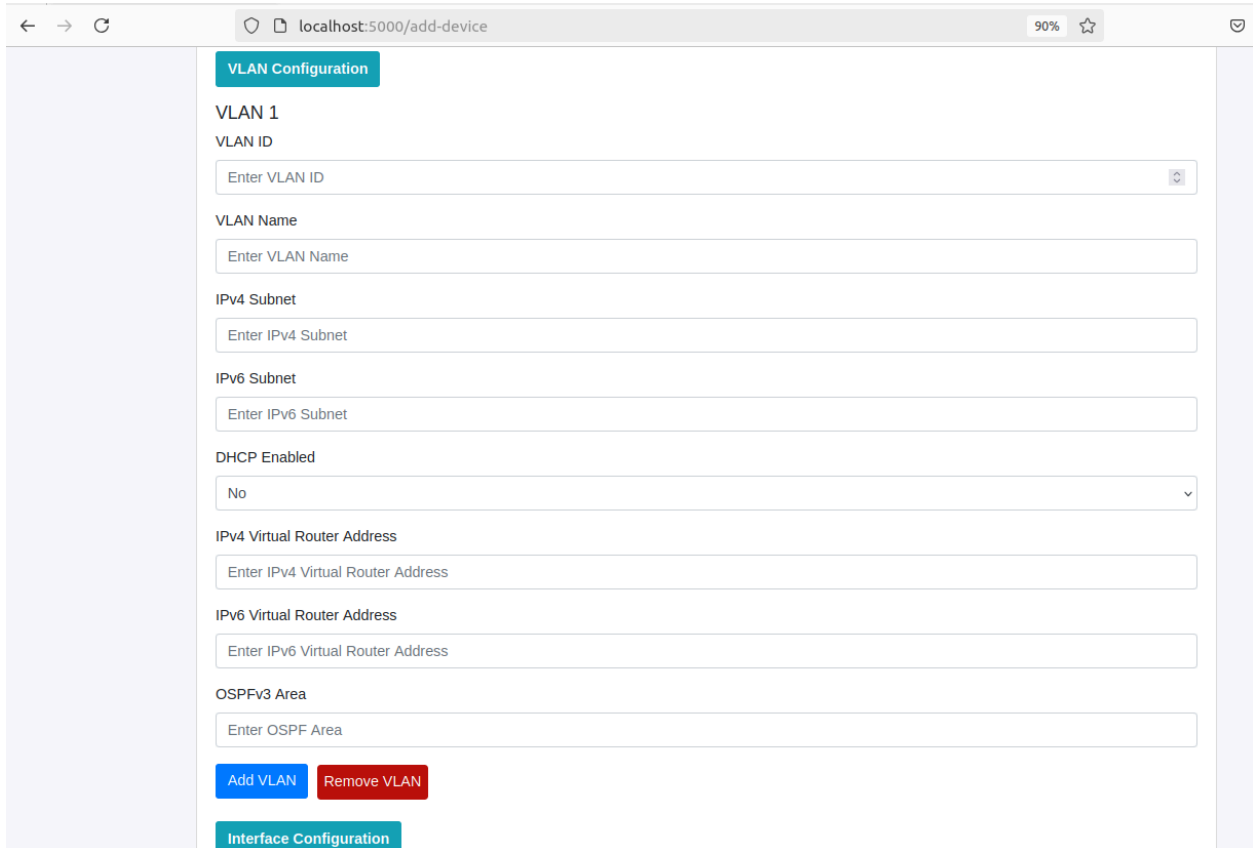
The screenshot shows a web browser window with the address bar displaying 'localhost:5000/add-device'. The form contains the following elements:

- Device Name:** A text input field containing 'R1'.
- Router Type:** A dropdown menu with 'Access' selected.
- Access Router Configuration:** A section with three blue buttons: 'VLAN Configuration', 'Interface Configuration', and 'Static Routes Configuration'.
- Static Route Prefix:** A text input field with the placeholder 'Enter Static Route Prefix'.
- Static Route Next Hop:** A text input field with the placeholder 'Enter Static Route Next Hop'.
- Add Static Route:** A blue button.
- Remove Route:** A red button.
- IPv6 Static Routes Configuration:** A blue button.
- OSPF Configuration:** A blue button.
- RIP Configuration:** A blue button.
- Submit:** A green button at the bottom.

*Screenshot 2: Collapsible sections of the form*

## Dynamic Form Interaction

JavaScript is used to toggle between the forms for different router types. Based on the selected router type, different configurations (*VLANs*, *interfaces*, *routing protocols*) are made available.



The screenshot shows a web browser window with the address bar displaying 'localhost:5000/add-device'. The page content is titled 'VLAN Configuration' in a teal header. Below this, the form is titled 'VLAN 1'. It contains several input fields: 'VLAN ID' with a placeholder 'Enter VLAN ID', 'VLAN Name' with a placeholder 'Enter VLAN Name', 'IPv4 Subnet' with a placeholder 'Enter IPv4 Subnet', 'IPv6 Subnet' with a placeholder 'Enter IPv6 Subnet', 'DHCP Enabled' with a dropdown menu currently showing 'No', 'IPv4 Virtual Router Address' with a placeholder 'Enter IPv4 Virtual Router Address', 'IPv6 Virtual Router Address' with a placeholder 'Enter IPv6 Virtual Router Address', and 'OSPFv3 Area' with a placeholder 'Enter OSPF Area'. At the bottom of the form section, there are two buttons: a blue 'Add VLAN' button and a red 'Remove VLAN' button. Below the form section, there is a teal header for 'Interface Configuration'.

Screenshot 3: Dynamic Forms

## Backend Overview

Upon submission, the form data is sent to the backend, which is handled by a **Flask** route (*/add-device*). Here, the form data is processed and stored as **YAML** configuration files.

The files are saved in a specific directory (*/home/student/git/csci5840/lab4/*), with a naming convention *{device\_name}\_{type of router}.yaml*.

For example:

- If the user adds a Core router named **R3**, the generated file will be: **R3\_core.yaml**.

- If the user adds an Access router named **R1**, the generated file will be: **R1\_access.yaml**.
- The files follow the YAML format and can easily be used for configuration generation.

```
student@csci5840-vm1-snir8112:~/git/csci5840/lab4$ ls | grep yaml
r1_access.yaml
r3_core.yaml
student@csci5840-vm1-snir8112:~/git/csci5840/lab4$ pwd
/home/student/git/csci5840/lab4
student@csci5840-vm1-snir8112:~/git/csci5840/lab4$
```

*Screenshot 4: Submitted form stored as a YAML file*

## Configuration Generation using Python

To streamline the generation of router configurations, I developed a Python script that uses **Jinja2** templates and **YAML** files to automate the process. This eliminates the need for manual configuration writing, improving efficiency and reducing errors.

### YAML and Jinja2 Integration

The script reads **YAML files** provided, which define the configurations for devices such as **r3\_core.yaml**. The appropriate **Jinja2 template** (e.g., *access.j2*, *core.j2*, *edge2.j2*) is selected based on the router type.

### Command-Line Argument Parsing

The script accepts two command-line arguments:

1. **--config**: The YAML configuration file (e.g., *r1\_access.yaml*).
2. **--type**: The type of router (Access, Core, Edge).

```
python generate_config.py --config r1_access.yaml --type access
```

Based on the **type** argument, the script selects the correct Jinja2 template and renders the configuration.

```
student@csci5840-vm1-snir8112:~/git/csci5840/lab4$ python3 generate_config.py -h
usage: generate_config.py [-h] --config CONFIG --type {access,core,edge}

Generate router configuration from YAML file.

options:
  -h, --help            show this help message and exit
  --config CONFIG        Path to the YAML file.
  --type {access,core,edge}
                        Type of the router (access, core, edge).
student@csci5840-vm1-snir8112:~/git/csci5840/lab4$
```

*Screenshot 5: Terminal Command Execution*

## Output

The output is a **fully rendered configuration** file based on the YAML data and the selected template.

```
student@csci5840-vm1-snir8112:~/git/csci5840/lab4$ python3 generate_config.py --config r1_access.yaml --type access
hostname r1

username admin privilege 15 role network-admin secret

management api http-commands
  no shutdown

daemon TerminAttr
  exec /usr/bin/TerminAttr -ingestgrpcurl=unix:/var/run/ingestgrpc.sock -taillogs --ingestauth=key,user:user --gnmi -grpc

vlan 10
  name h1-h3

dhcp server

  subnet 11.0.0.2/24
    range 11.0.0.10 11.0.0.50
    name h1-h3
    default-gateway 11.0.0.3

  !

  subnet 2010::2/64
    range 2010::2 2010::ffff:ffff:fffe
    name h1-h3
```

*Screenshot 6: Generated Router Configuration Output*

## Saving configs

To ensure regular and automated backups of network configurations, I implemented a systemd service ([backup-configs.service](#)) that runs a Python script to collect and store the

running configurations of routers. The service automatically connects to each router, retrieves the configuration, and saves it in `/home/student/git/csci5840/cfgs` directory.

## Service Setup

The systemd service is defined to run continuously

```
student@csci5840-vm1-snir8112:~/git/csci5840/lab4$ systemctl status
backup-configs.service

backup-configs.service - Backup Network Configs Service
   Loaded: loaded (/etc/systemd/system/backup-configs.service; enabled;
   vendor preset: enabled)
   Active: active (running) since Mon 2024-09-30 19:55:19 MDT; 59min ago
 Main PID: 12084 (python3)
    Tasks: 161 (limit: 4915)
   Memory: 39.5M
      CPU: 3min 19.600s
   CGroup: /system.slice/backup-configs.service
           └─12084 /usr/bin/python3
/home/student/git/csci5840/scripts/backup_configs.py
```

*Screenshot 7: Service status*

```
student@csci5840-vm1-snir8112:~/git/csci5840/cfgs$ ls
r1.cfg r2.cfg r3.cfg r4.cfg r5.cfg s1.cfg s2.cfg s3.cfg s4.cfg
```

*Screenshot 8: Config files for each router*