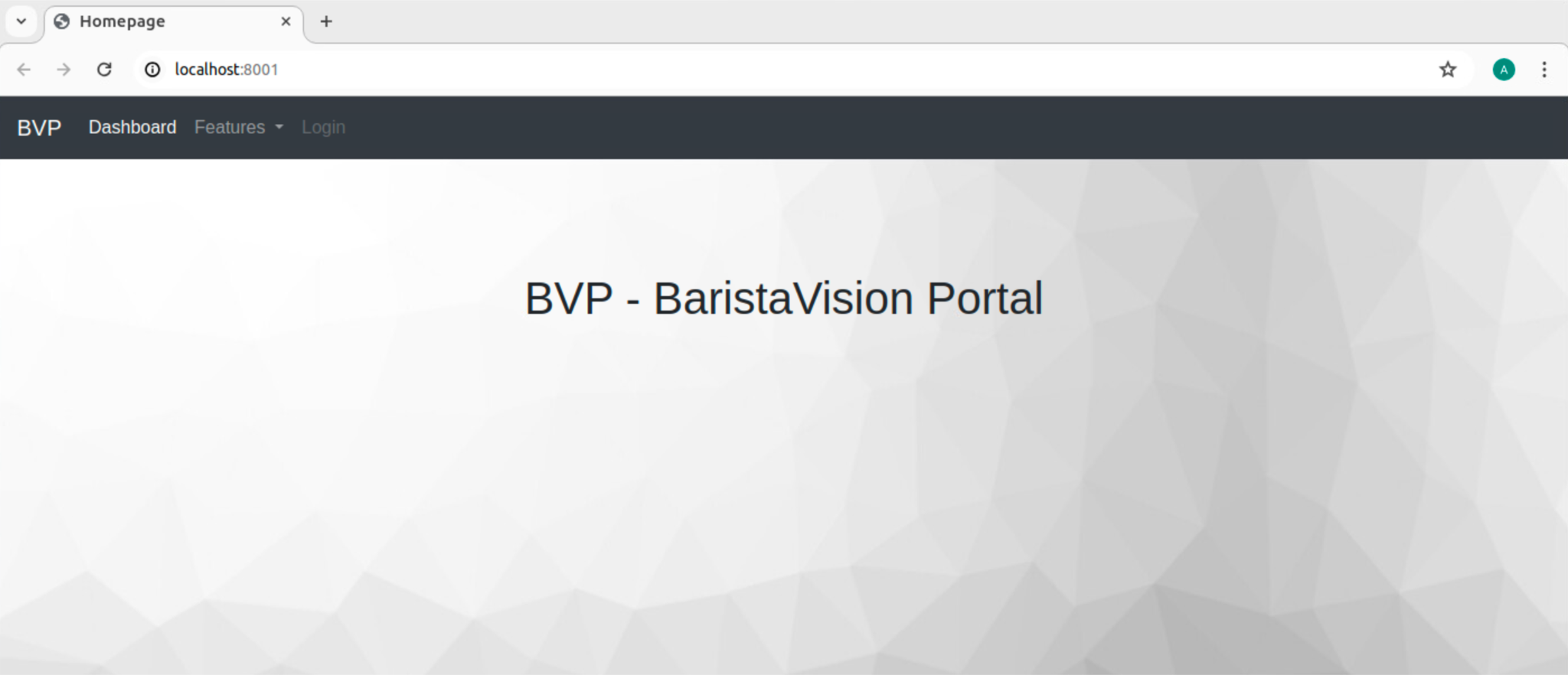
**1. Frontend**

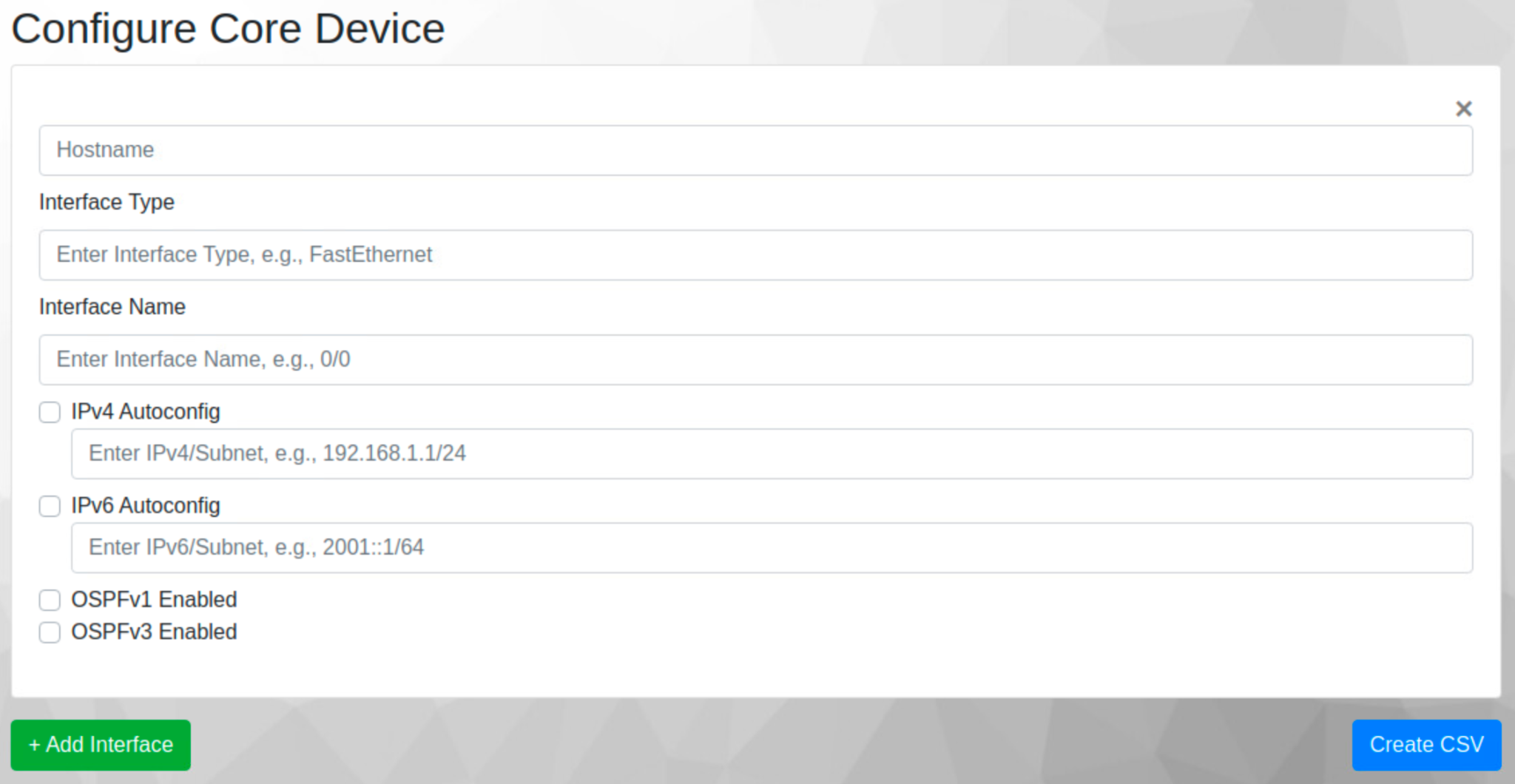
**Introduction**

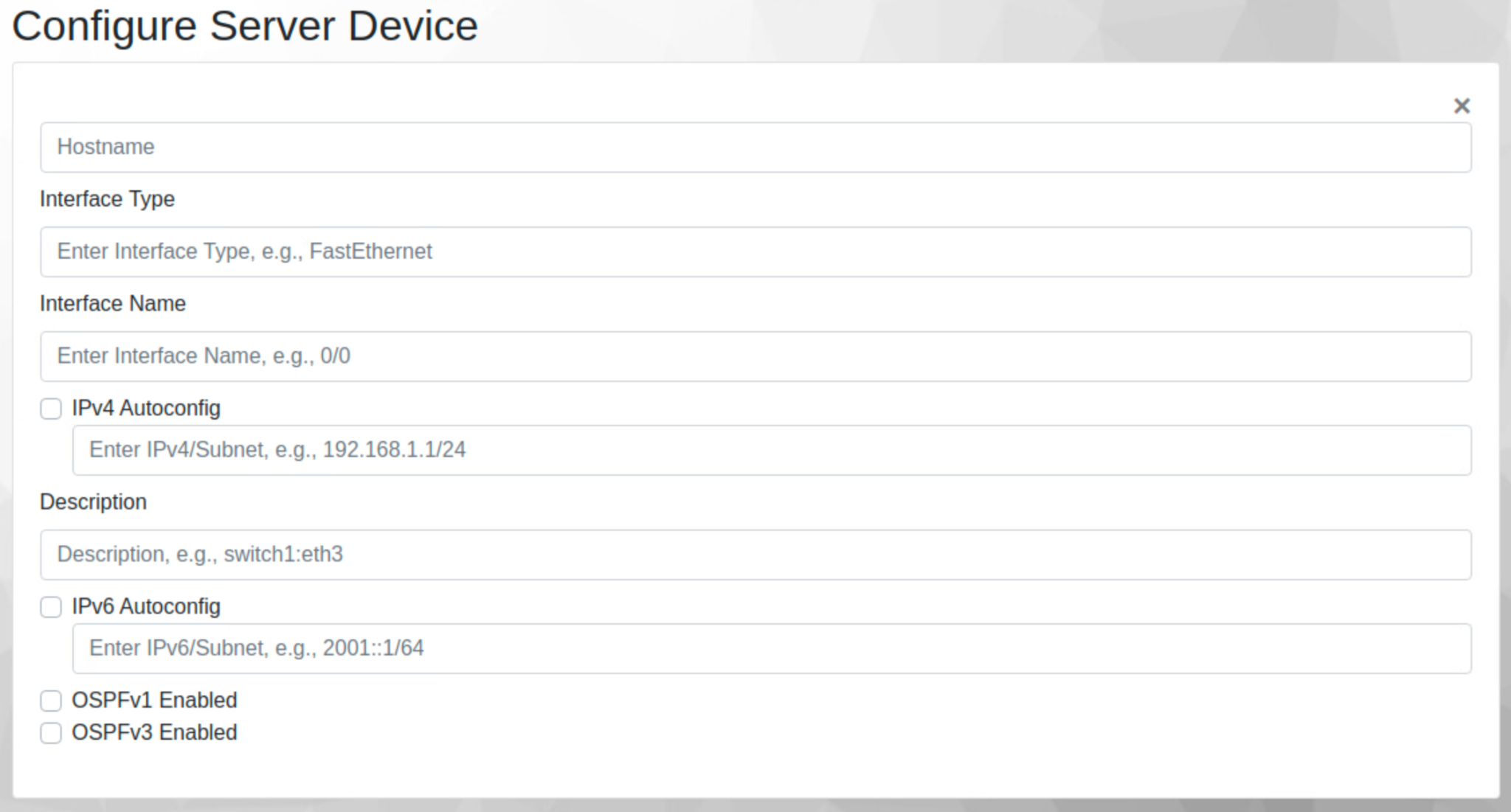
This section focuses on the front end of our web application BaristaVision Portal (BVP). BVP is an end-to-end network Automation tool that helps WDTC to configure their entire backbone network on the go. Abstraction is very important when it comes to automation tools where the end user’s minimal inputs are efficiently processed in a way that the networking devices understand. A front end GUI which contains sophisticated information in a human readable format is the goal of BVP.

* The tool follows the industry standard for configuring and monitoring devices by logically categorizing the devices based on their role within the network. They are categorized as Edge devices, Core devices, Internal devices and Server devices. Dedicated app routes for these devices are present. Thanks to Flask. This information is stored in the form of CSV which is easy to read and process in our upcoming stages.





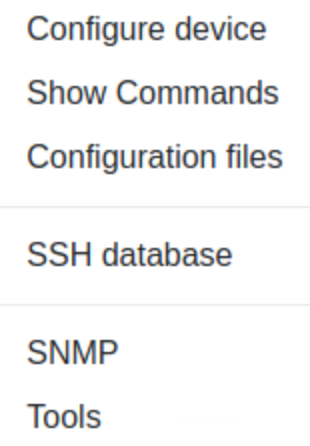




* These user inputs are then processed and stored as csv which is then uploaded to the GitHUB. This is a very crucial step as we follow a structured templating using Ansible, Jinja, Netmiko all of which are triggered by the jenkins as we push our csv file to the git repository.



Incorporated basic tools making it easier for end users to get information within a few clicks.



**Conclusion**

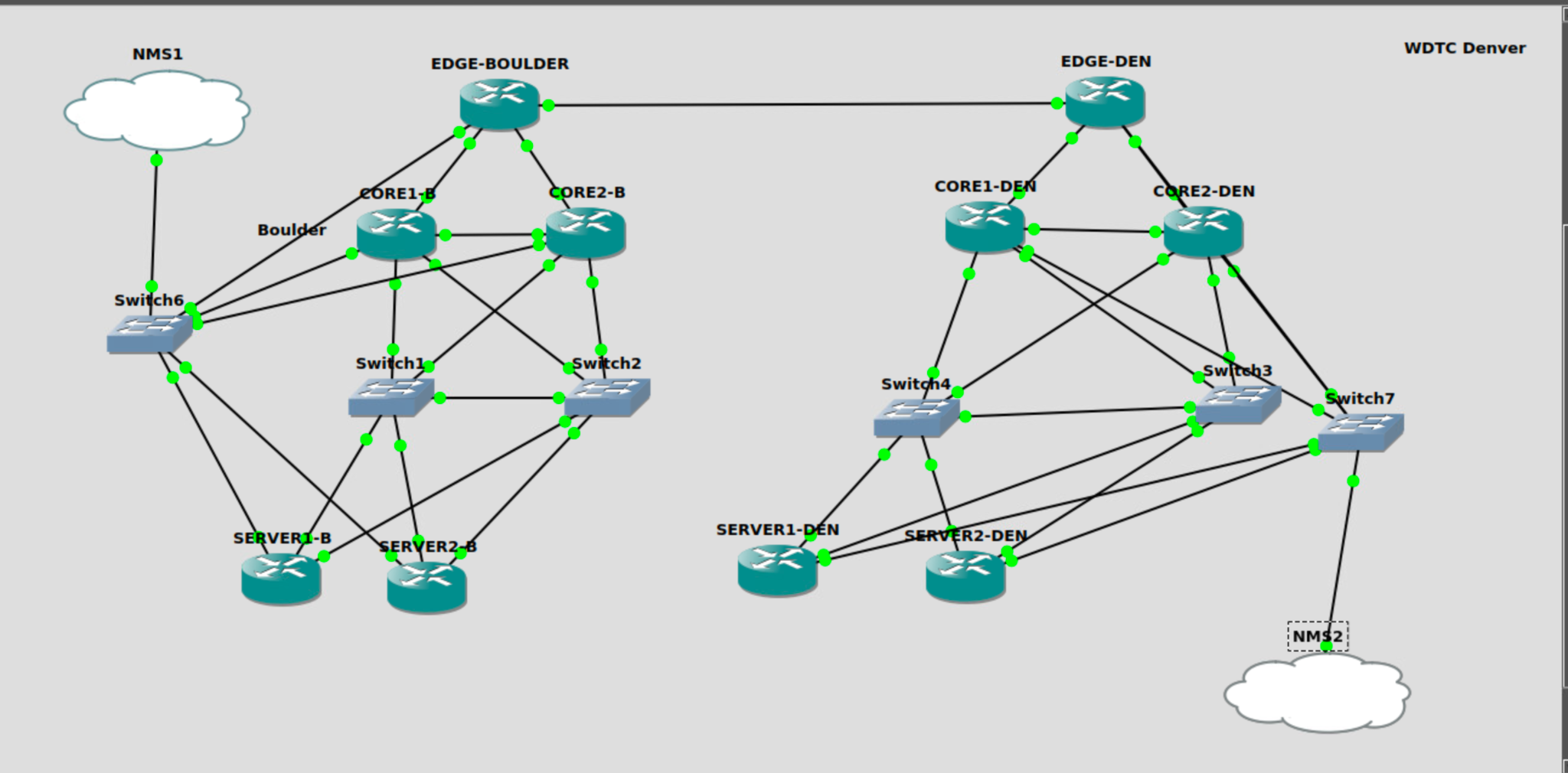
The solution we provide is valued by how useful it is to the customers than that of it simply being a perfect solution. Keeping this in mind we worked closely with the entire team in creating a powerful yet easy to access tool. We present you BVP.

**2. Backend**

This backend section of the report focuses on the amazing automation efforts here at Barista. The tools we utilized to implement the backend automation are: Netmiko, Ansible-YAML-Jinja2, .csv management and Jenkins. In this section I will go over how each aspect of the automation framework works.

Our solution aims to provide a centralized network management system that can automate any section of your new network. In this infrastructure, everything is standardized and can be scaled up to any number of devices at each layer including: EDGE, CORE, and SERVER. Below is a representation of the new Boulder data center and the current Denver site data center. We used this diagram to illustrate the proof of concept using a fraction of the devices you will utilize for the real implementation. While this may seem small, it can easily scale to any amount of devices

**Network Diagram:**

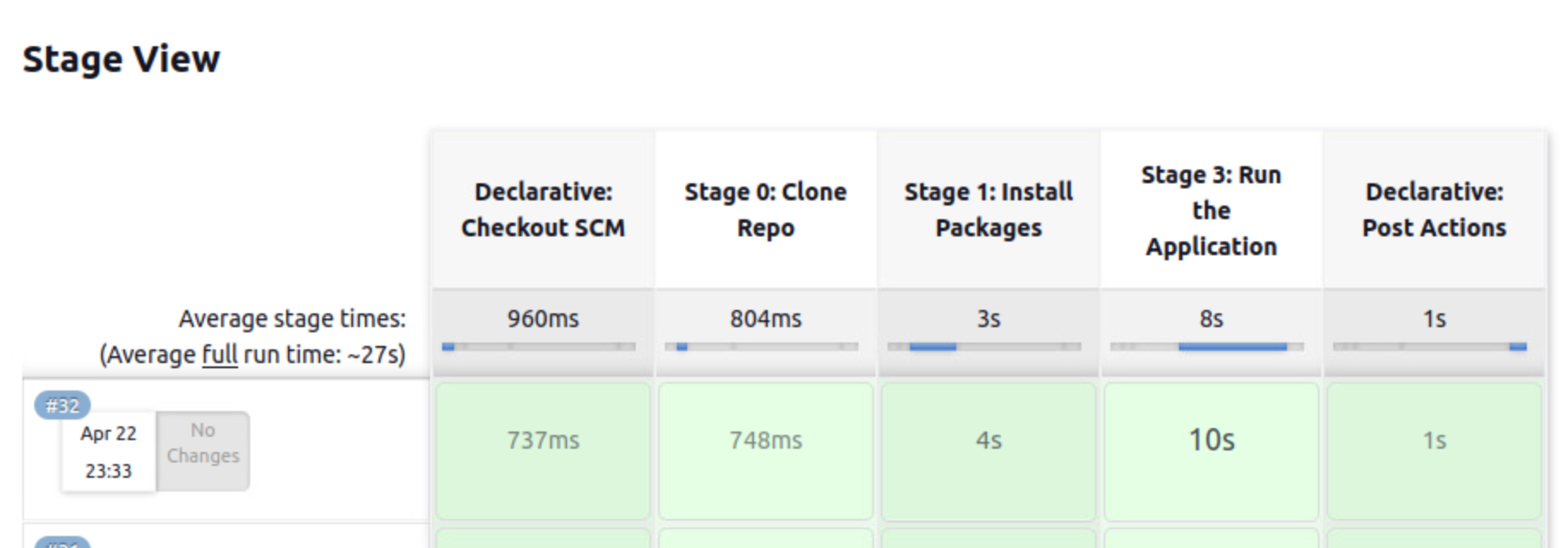
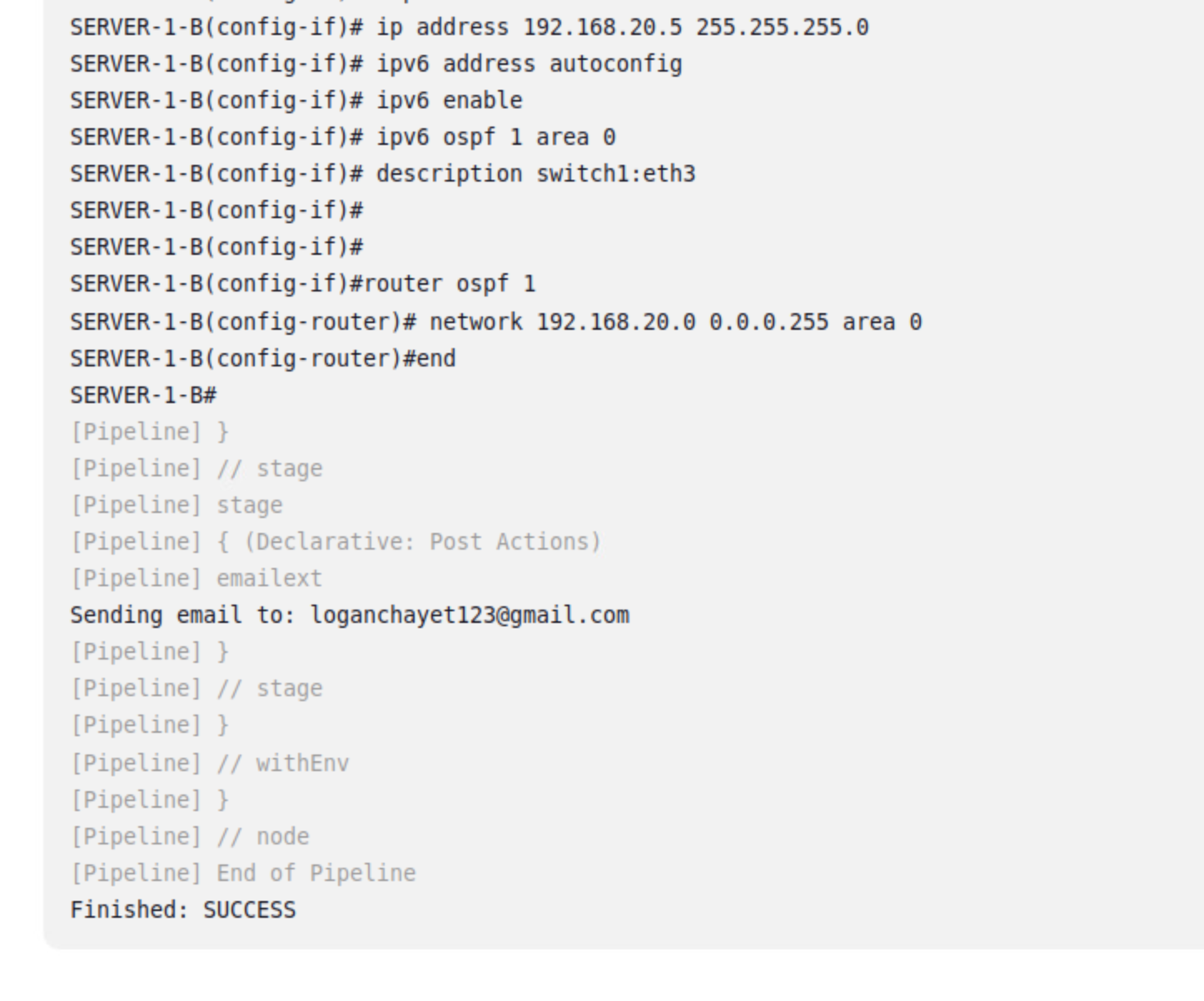


**The Setup:**

1. **Jenkins**

We included Jenkins in our setup to encourage change management and increase the ease of use through the front end. Our Jenkins instance is connected to a GitHub repository that it will pull information from that is triggered via a github webhook. Once the front end adds a configuration, it will create a .csv that pushes that .csv to the github. This will then trigger the Jenkins job. The Jenkins job checks for certain libraries that are installed so that it can run the python files. Then it will run the playbookCreation.py which will execute all the Ansible automation code and push via Netmiko. Whether the job fails or succeeds, it will send an email to the desired person as configured in Jenkins. If need be, a CRON job can be configured to push configs automatically at a certain time.

Here is an example of a successful Jenkins job:

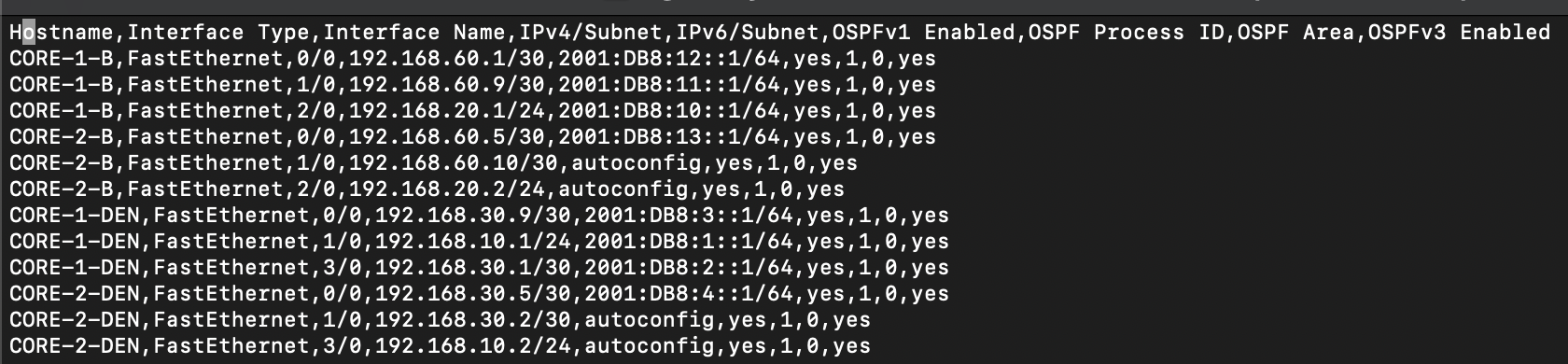


The Jenkins file is provided below:



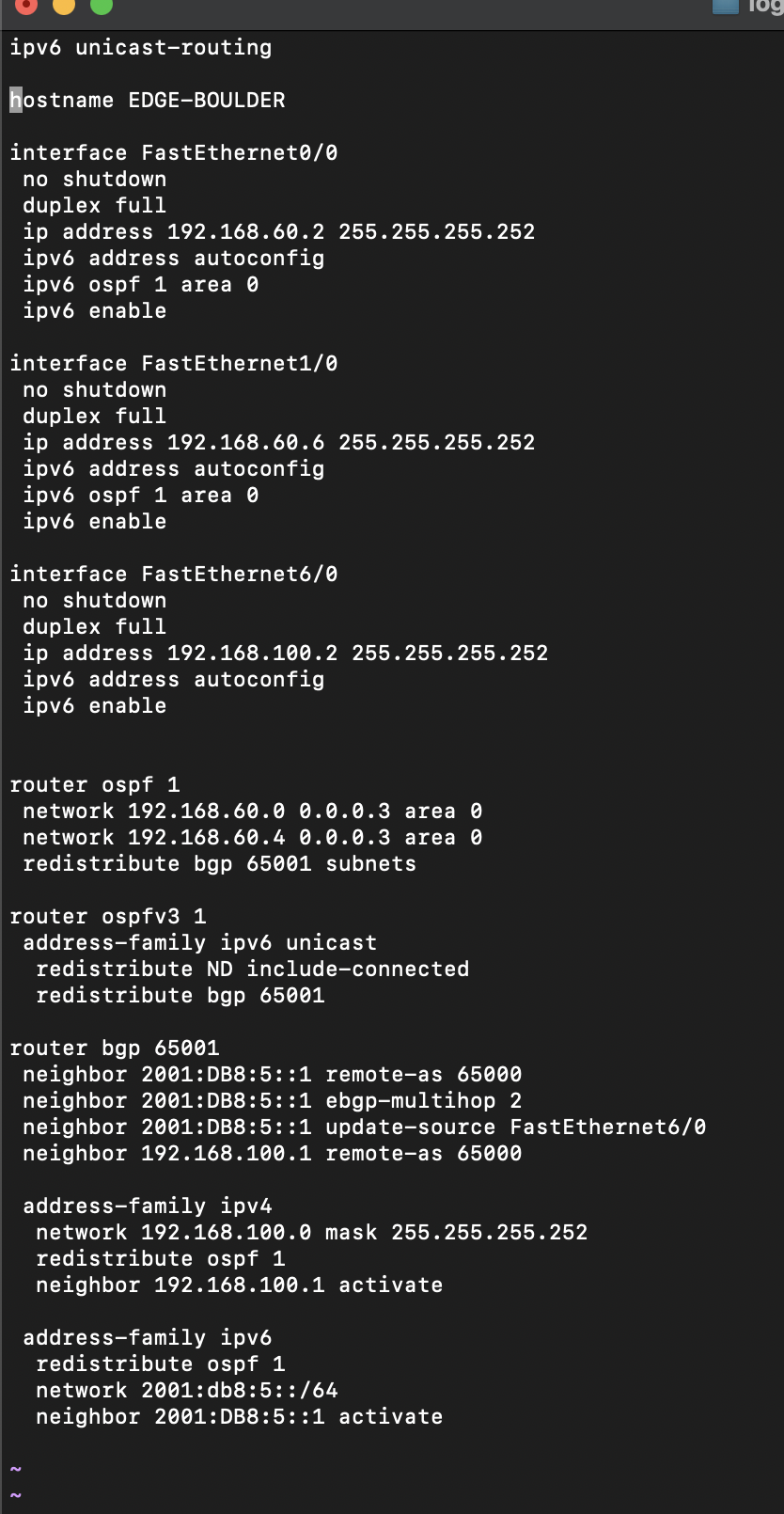
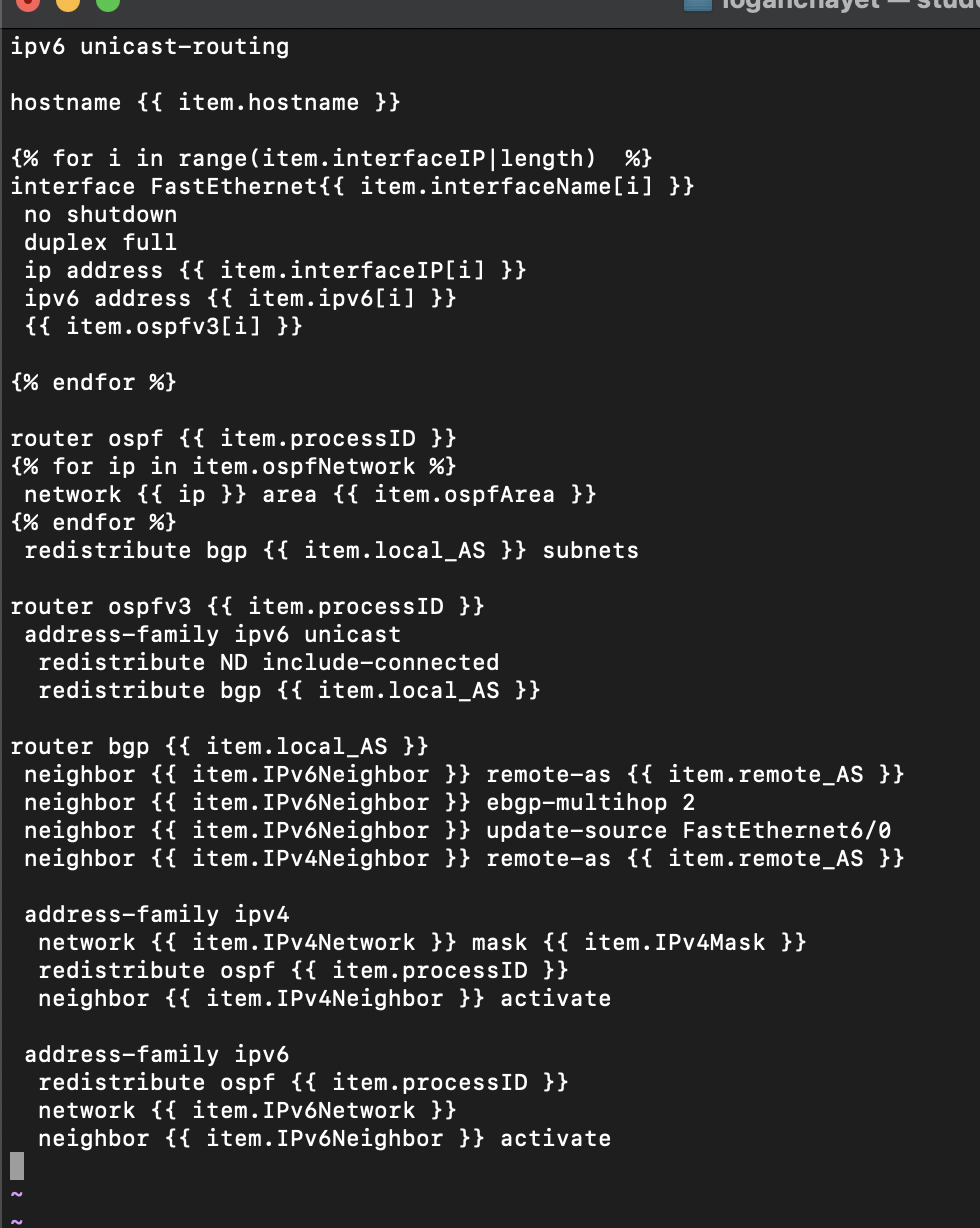
1. **Ansible-YAML-Jinja2**

We utilized Ansible to create configurations with Jinja2 templates. First, we have a playbookCreation.py file that executes all the functionality to pull data from a config\_requirements.csv to push it to the devices. First, we have a createPlaybook() function for each type of router: EDGE, SERVER, and CORE. This creates a .yaml playbook that inserts the file into the Ansible structure (similar to Lab 7 - Ansible) based on the information provided from the config\_requirements.csv (example below).

****

When the Ansible .yaml file is create, the .py file will then run the command “ansible-playbook site.yaml –tags edge” which will then take the Jinja2 template that is stored in that specific router role and create a “EDGE-BOULDER.txt” configuration file. After this occurs, Netmiko will trigger and send the configurations to the routers.

Here’s an example of a Jinja2 file for the EDGE routers and what the config output looks like:



Finally, all new configurations will be stored locally on the NMS for future reference as golden configurations.

**Conclusion:**

This backend automation allows for your network to be not only scalable, but trackable with change management and email notifications to allow for the user to fully understand what is going on with the current build.

**3. Monitoring**

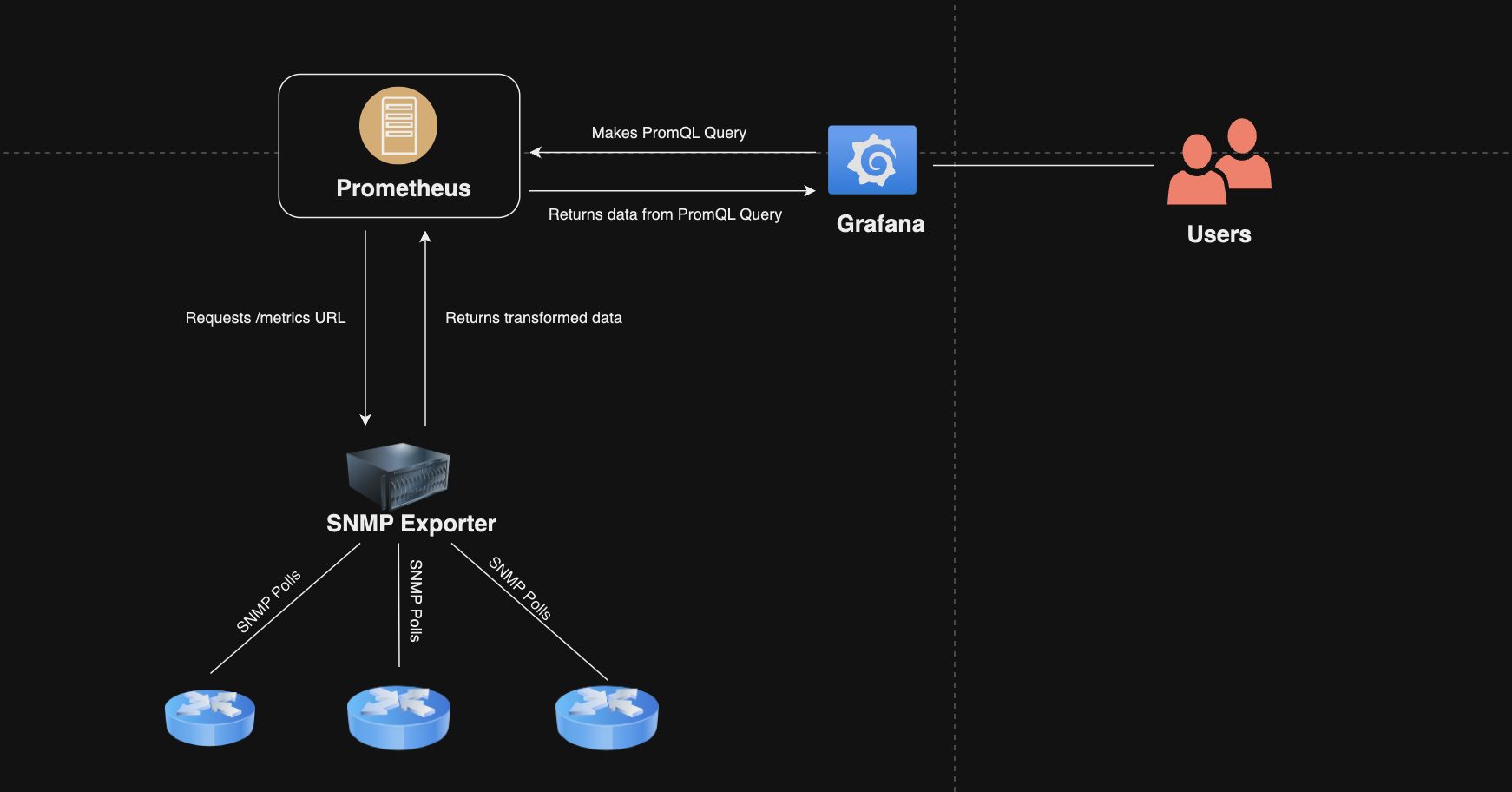
**Introduction**

This report presents a comprehensive solution for monitoring network devices within two key data centers located in Boulder and Denver for WDTC. Leveraging SNMP Exporter, Prometheus, and Grafana, this report demonstrates how your organization can implement robust monitoring systems to gain insights into network performance, identify potential issues proactively, and ultimately optimize the reliability and efficiency of their data center operations.

Our solution aims to provide a centralized monitoring system hosted on a server. It can identify the health of all the network devices at one single location and can alert if there’s any certain threshold has been reached. You also have everything accessible in one single UI that is fully customizable. We can now easily identify bottlenecks on the network.

Monitoring comes in 2 different forms: Logs and Metrics. Here, we'll focus on Metrics. Metrics provide statistical data such as MTUs, Network Traffic, and interface status, offering a systemic view of system performance and behavior over time. Unlike logs, which offer detailed event-level information, metrics enable trend analysis, proactive problem identification, and optimization of resource utilization.

**The Architecture**

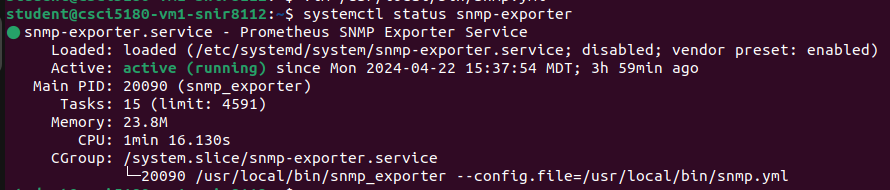


**The Setup**

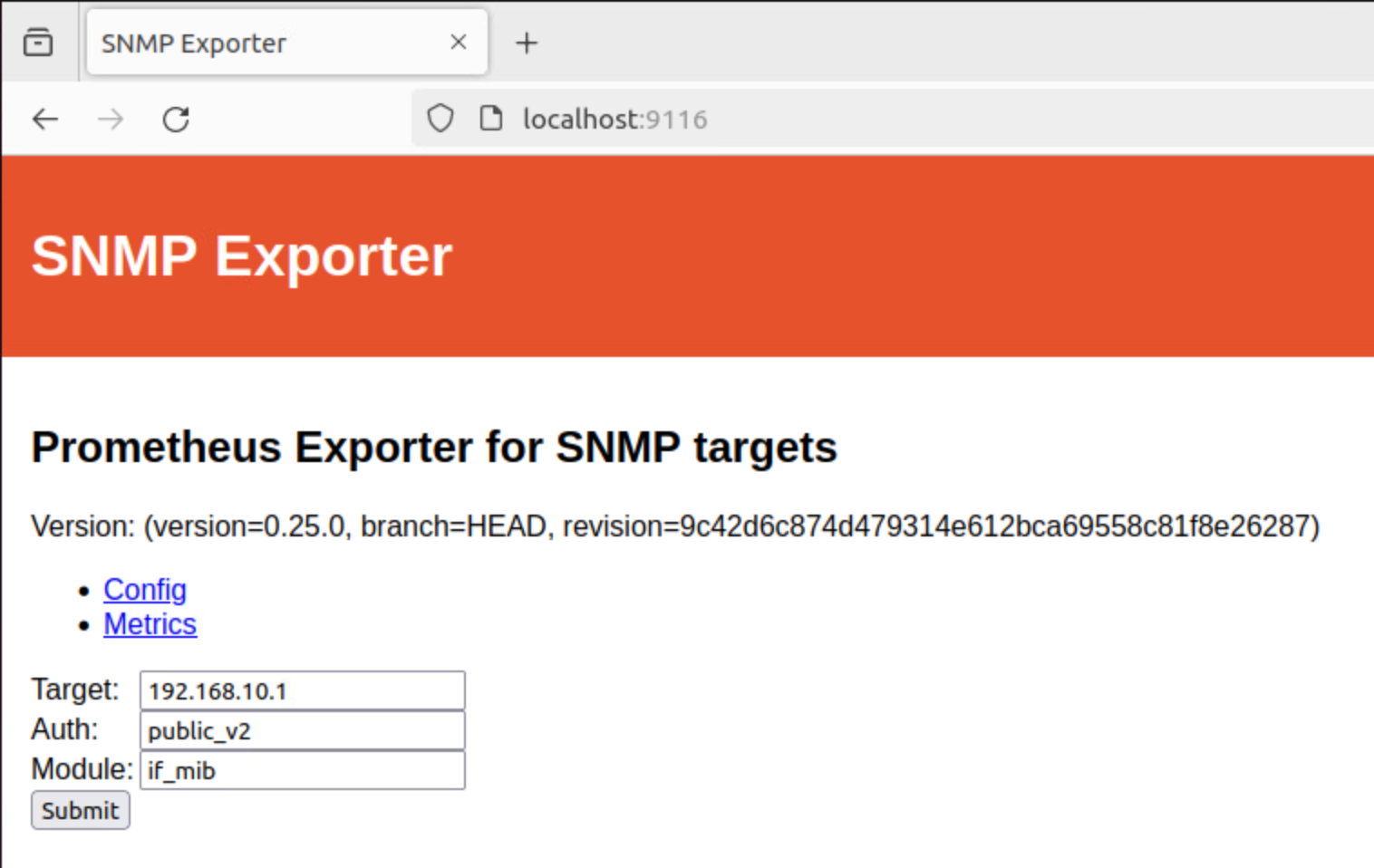
1. **SNMP Exporter**

SNMP Exporter is a tool that collects data from devices such as routers, switches by querying their SNMP agents. SNMP Exporter then exposes this data in a format that can be scraped by monitoring systems like Prometheus. It allows for the retrieval of various metrics such as network traffic, memory utilization, and interface status from SNMP-enabled devices.

We have the SNMP Exporter service running on our NMAS, which is crucial for gathering SNMP metrics.



The SNMP Exporter service utilizes a configuration file named **snmp.yml**, containing all the necessary Object Identifiers (*OIDs*) for metric collection. Furthermore, the SNMP Exporter service runs on localhost at port 9116, making the collected metrics accessible for monitoring and analysis.

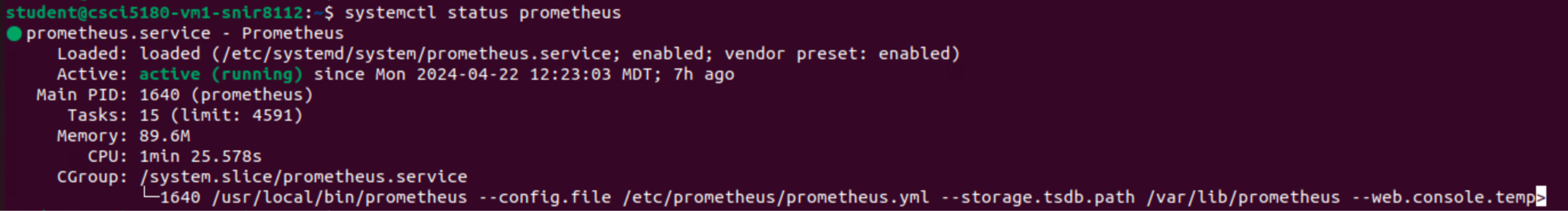


SNMP Exporter transforms SNMP data into **Prometheus** metrics by querying the specified OID data from the Agent and mapping the data to readable metrics.

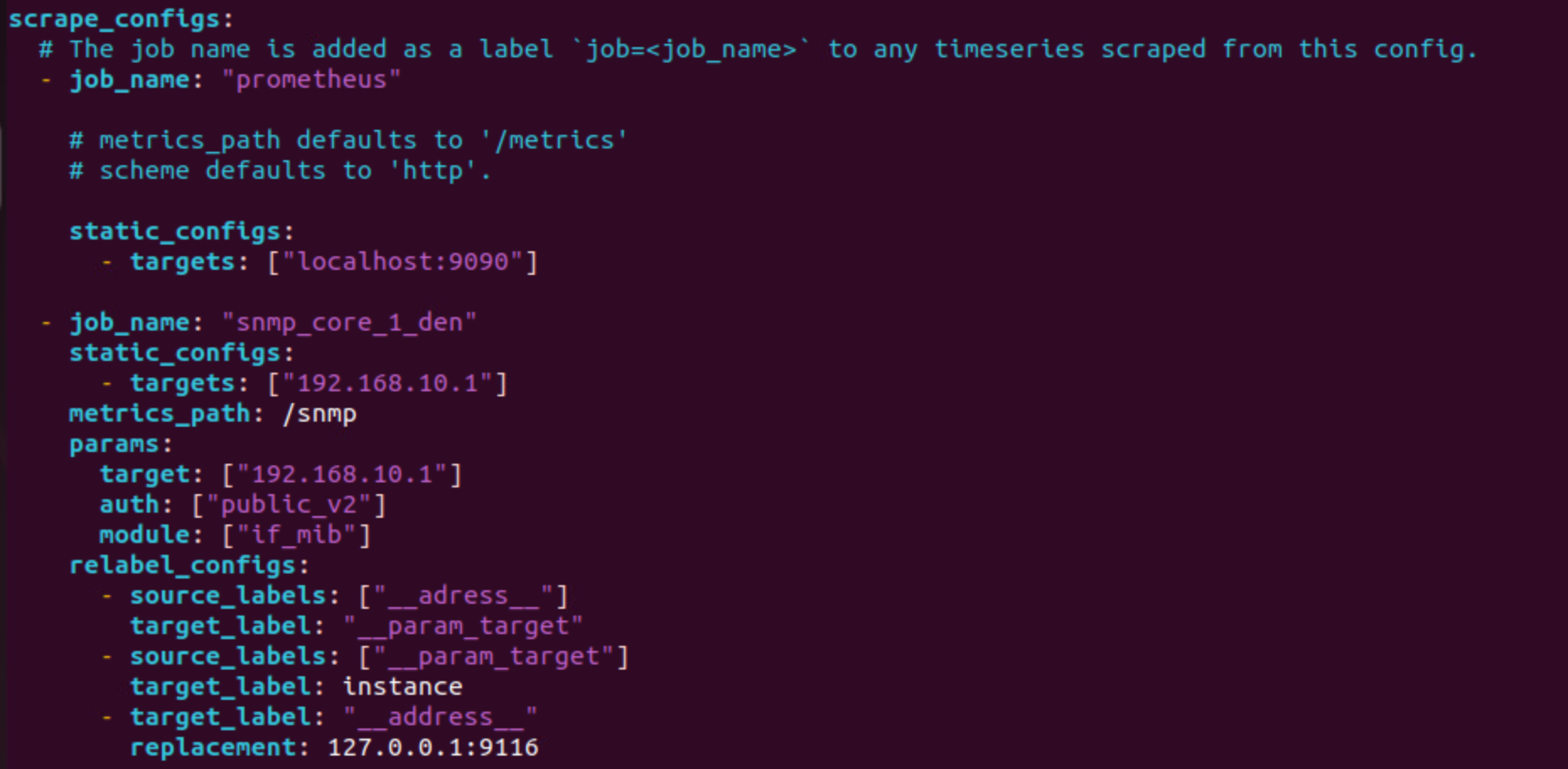
1. **Prometheus**

Prometheus is an open-source monitoring tool that collects time-series data from various sources (*snmp exporter in our case*) using a pull-based model. It stores this data in a time-series database and provides a powerful query language, PromQL for analyzing and visualizing metrics.

We have the prometheus server running on our NMAS.

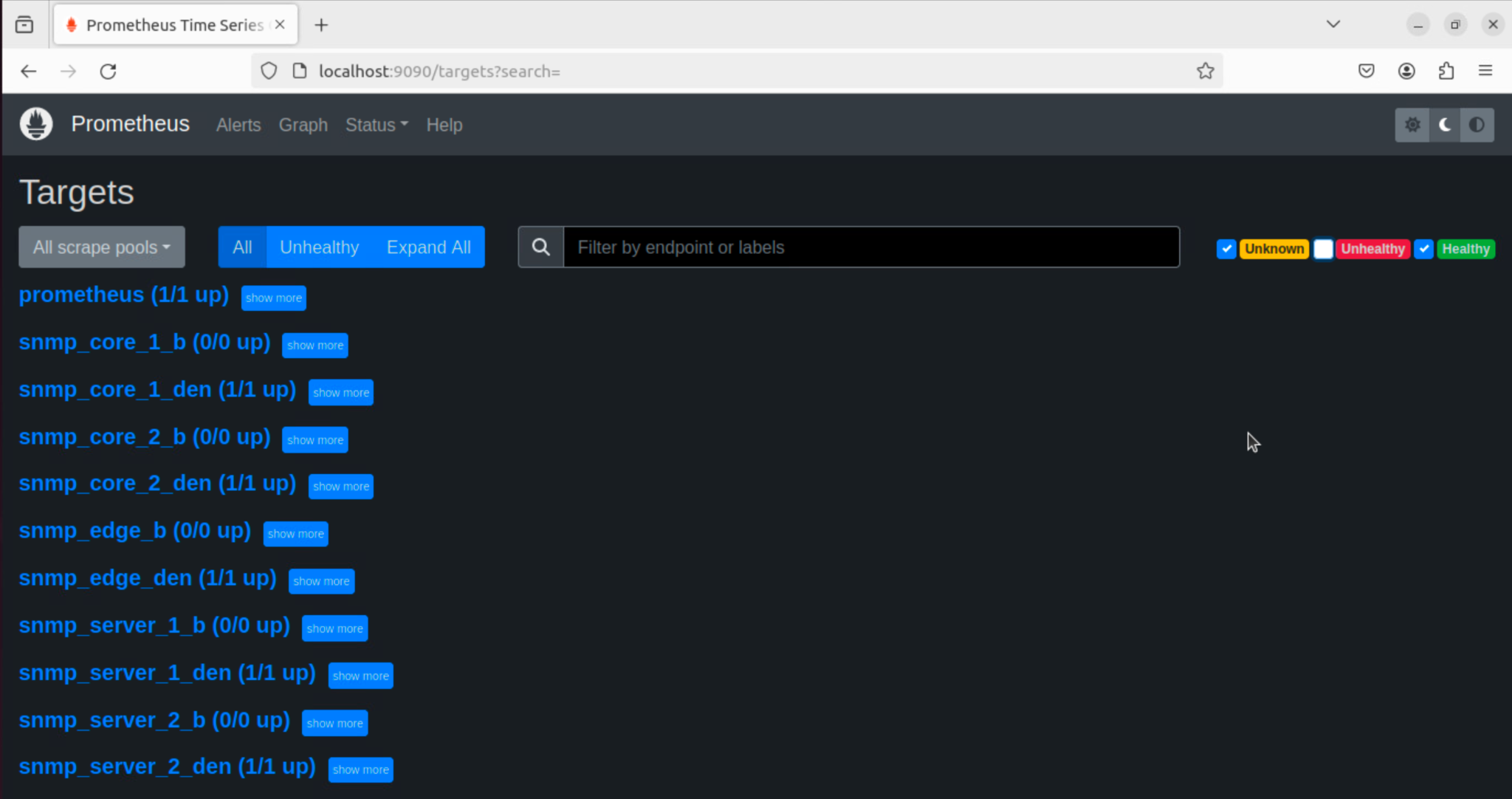
****

Here’s a snippet of prometheus.yml file.

****

* This configuration collects metrics from Prometheus itself, targeting *localhost:9090,* where the Prometheus server would run on.
* It collects SNMP metrics from a specific device (*192.168.10.1*) using SNMP Exporter. We have all the other routers configured as separate “*targets*” in the yaml file.
* It also specifies the SNMP community string for authentication as "*public\_v2*" and the SNMP module to use as "*if\_mib*".
* Additionally, it relabels target labels to ensure proper identification and replaces the target address with the SNMP Exporter address (*localhost:9116*). This configuration is crucial since Prometheus redirects all data to our SNMP Exporter which would transform the data in a query that Prometheus would understand.

A snippet of all the “targets” or devices on Prometheus.

****

With Prometheus set up, we leveraged Grafana's robust visualization capabilities. While Prometheus efficiently collects and stores SNMP metrics, Grafana complements this functionality by offering intuitive dashboards and customizable visualizations for monitoring and analyzing network performance metrics.

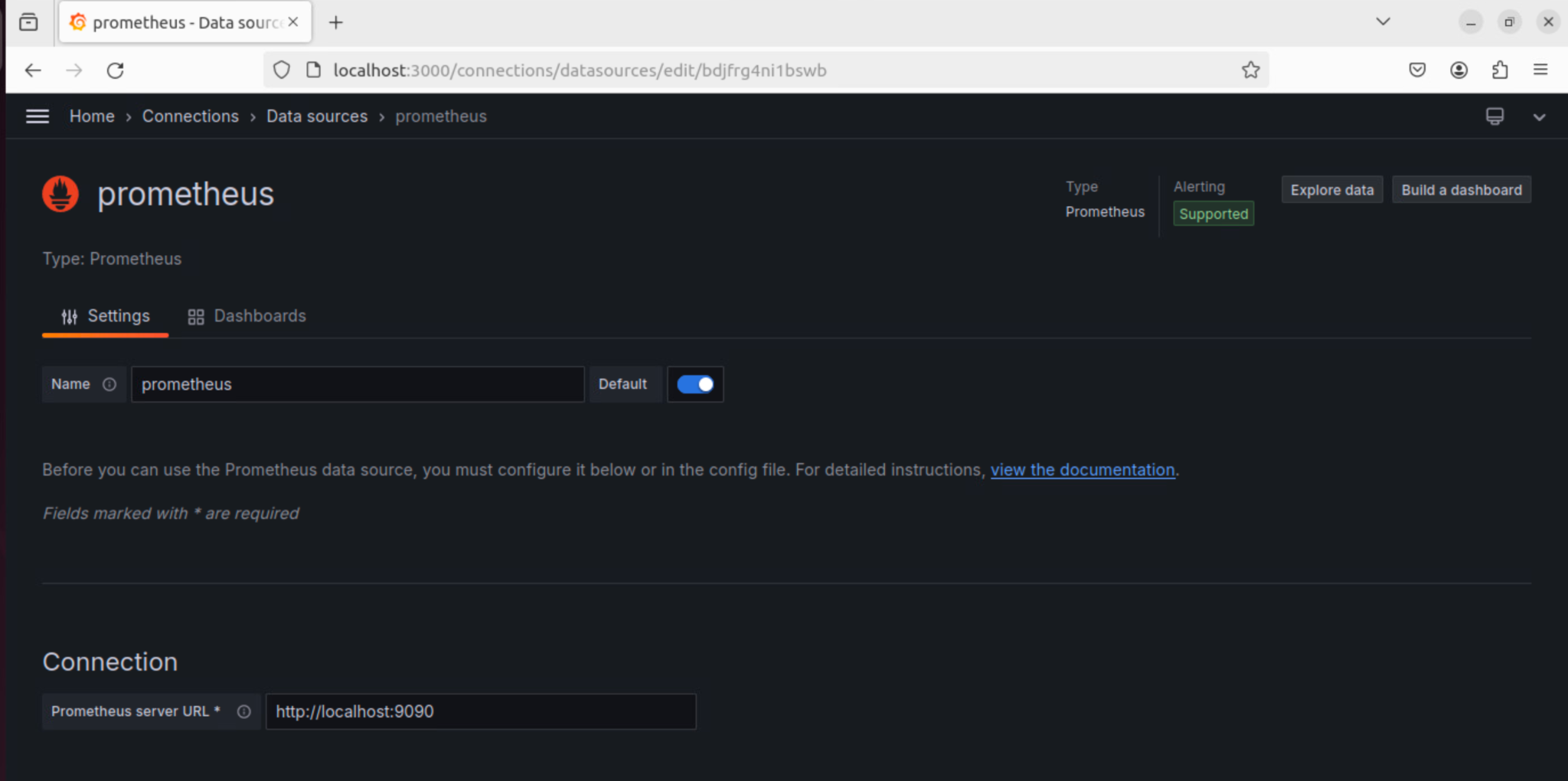
1. **Grafana**

Grafana is an open-source analytics and visualization platform designed to help users understand and monitor their data. It provides a highly customizable and feature-rich environment for creating interactive dashboards, charts, and graphs, allowing users to visualize and analyze data from various sources in real-time.

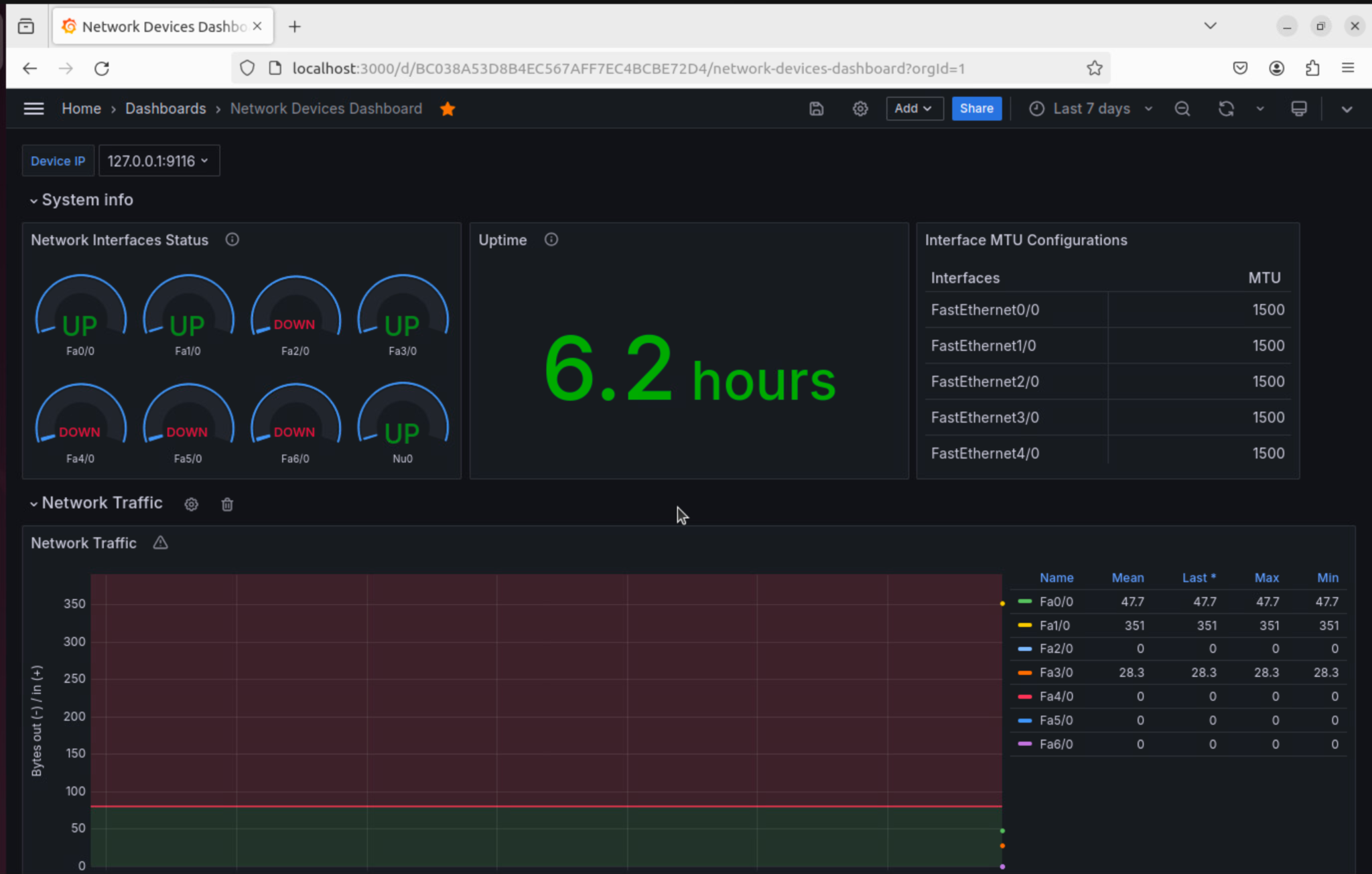
We have the Grafana server hosted on our NMAS.

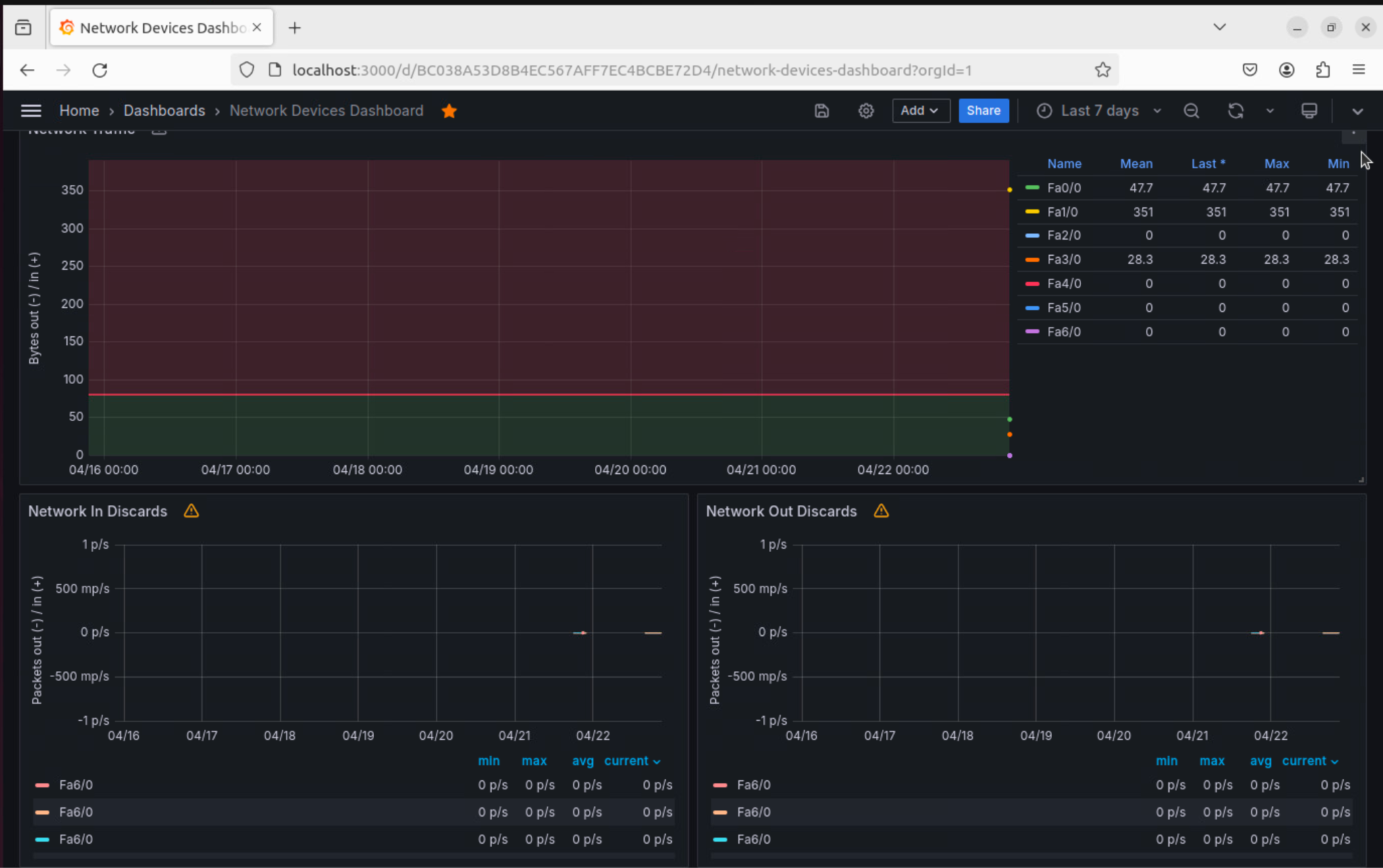


We configured Grafana’s data source to be Prometheus so that it pulls data from the Prometheus server we configured on localhost:9090.



We created a comprehensive “**Network Devices Dashboard**” that visualizes all the data retrieved from Prometheus. Here’s a snippet of the how the dashboard looks like.





Finally, we have all the required Prometheus metrics visualized on Grafana!