

MANAGEMENT OF NETWORK DEVICES USING THE RESTCONF PROTOCOL

Cornelia NIȚĂ, Adrian MIHĂILEANU, Vlad VASILE, Annamaria SÂRBU

“Nicolae Bălcescu” Land Forces Academy, Sibiu, Romania

cornelianita2.0@gmail.com

Abstract: Modern network infrastructures are characterized by the need for accessing the configuration and operational data across multiple devices, an inefficient task when performed manually. This paper proposes a read-only solution for automated data extraction from network devices. Based on RESTCONF (Representational State Transfer Protocol), this method aims to reduce manual intervention and improve monitoring efficiency. Structured data is retrieved from routers deployed in a virtualized environment, filtered for relevant information then centralized through a web interface. The organized structure of the web page allows an intuitive presentation of the extracted data and rapid access to device data. With automated data collection and centralized visualization, this solution enables efficient monitoring while minimizing the risk of unintended configuration changes.

Keywords: Network Automation, RESTCONF, GET, Data Extraction

1. Introduction

Network device management has become a demanding process in contemporary network infrastructures [1]. Devices such as routers and switches must be supervised and monitored continuously in order to ensure reliable operation and network stability. As network architectures are experiencing ongoing evolution, they become more complex, increasing both the number of devices and the volume of data [2].

In such contexts, network administrators should regularly verify device configuration and operational status to ensure proper network performance. Traditional management relies on manual interaction with individual devices, in most cases through command-line interfaces. This approach is effective in small-scale environments, but it becomes time-consuming and prone to human error when it comes to more complex architectures [3]. To mitigate these limitations, modern network administration strategies are based on automation [4]. With the help of structured data representations and

standardized communication protocols, centralized access to devices is enabled. These challenges emphasize the need for standardized and centralized management mechanisms supported by protocols such as RESTCONF [5].

Network device management protocols are based on sets of rules that allow administrators to remotely access and manage network devices or services [6]. To simplify integration with different applications, RESTCONF was developed. It offers a simpler and more accessible approach by using the principles of RESTful architecture. This is a standard protocol that provides a programmatic interface for accessing data defined in the YANG (Yet Another Next Generation) language model, using the datastore concepts defined in NETCONF (NETwork CONfiguration protocol) [7]. A RESTCONF communication model between a management application and a network device is illustrated in Figure 1.

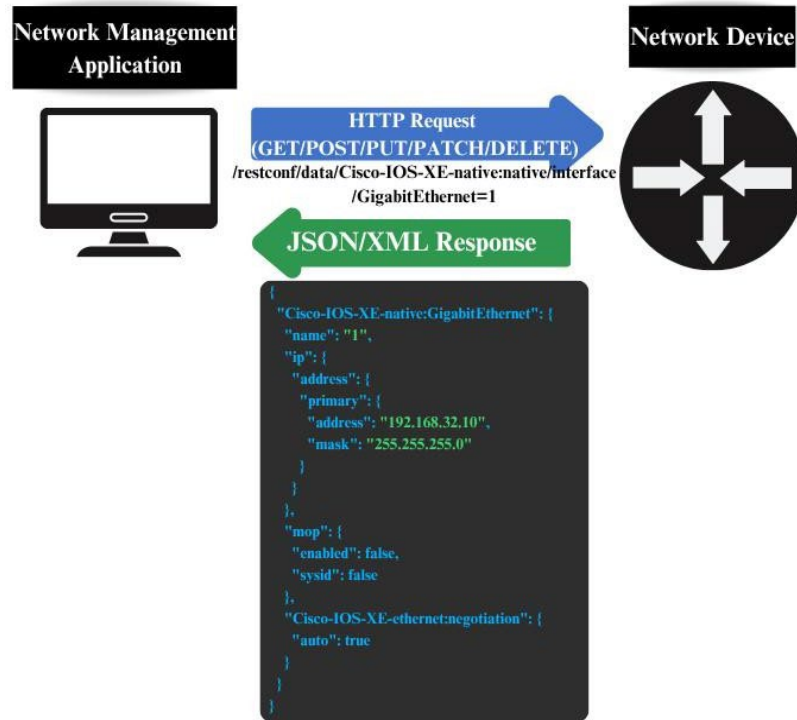


Figure 1: RESTCONF communication model

RESTCONF uses HTTP (Hypertext Transfer Protocol) and HTTPS (HTTP Secure) to manage configuration and state resources of network devices [8]. In this study, the GET operation is used as a read-only method to retrieve data from the devices.

In RESTCONF, the configuration and state data are described by YANG model [9]. This is a modelling language that defines the structure and type of resources that a server can manage.

The data formats supported by RESTCONF are JSON (JavaScript Object Notation) and XML (Extensible Markup Language). JSON is easier to read and process both by humans and machines. It is preferred because of the compact format and reduced resource consumption. JSON is also compatible with programming languages such as Python and Java.

2. RESTCONF-Based Management System Implementation

2.1. Materials and Methods

The implementation environment was created in GNS3 defining the physical and logical topology of the network. The

network is composed of three Cisco CSR100v (Cloud Services Router) devices, as illustrated in Figure 2.

VMware Workstation Pro was used as a support for GNS3 VM and also to integrate three separate virtual machines, each associated with a network device, responsible for its management. The data processing script and the web page were developed and implemented with the help of Visual Studio Code. The NGINX server was used to host the processed data files of the devices in order to be delivered to the web interface. The implementation relies on the following programming languages: HTML (Hypertext Markup Language), CSS (Cascading Style Sheets) and JavaScript for creating the web page and Python for creating a data processing script.

The methodology adopted in this study relies on the use of the RESTCONF protocol to automatically extract the configuration and operational status data from network devices.

The routers were accessed through RESTCONF API endpoints over HTTPS.

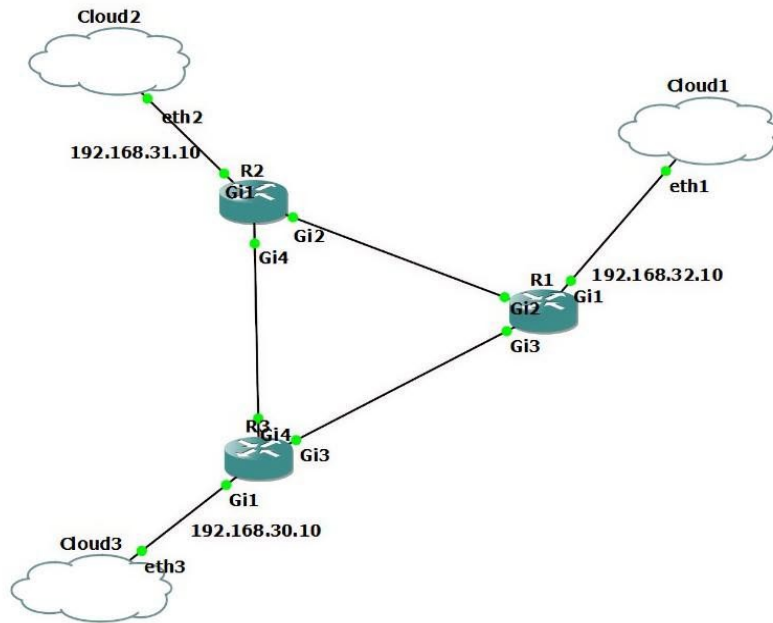


Figure 2: Network Topology used in the experimental environment

The responses were returned in JSON format and filtered to extract only the relevant information for this application. This solution uses Python libraries such as *requests* for HTTP communication and *json* for processing the retrieved data. The RESTCONF requests target YANG-based resources associated with interface configuration and operational status, while authentication was implemented using HTTP Basic Authentication. HTTPS

ensured encrypted communication between the management scripts and the network devices, while access to the web interface was restricted to the local virtual environment used for testing purposes. Interaction with the equipment was carried out through GET operations, ensuring read-only access. The data was, afterwards, processed to extract the information needed. The filtering stage implemented in Python is illustrated in Figure 3 through a

```

with open(input_file, 'r') as data:
    info = json.load(data)

router_info = {
    "hostname": info.get("Cisco-IOS-XE-native:native", {}).get("hostname", "Necunoscut"),
    "version": info.get("Cisco-IOS-XE-native:native", {}).get("version", "Necunoscuta")
}

interfaces = info.get("Cisco-IOS-XE-native:native", {}).get("interface", {})
all_interfaces = []
tunnel_interfaces = []

if isinstance(interfaces, dict):
    for key, value in interfaces.items():
        for interface in value:
            if "Tunnel" in key:
                tunnel_interfaces.append(interface)
            else:
                all_interfaces.append(interface)
else:
    all_interfaces = interfaces

```

Figure 3: Python code fragment for RESTCONF JSON filtering

representative code fragment that shows how the JSON response is parsed and how physical and tunnel interfaces are separated for further analysis.

The code fragment shows how the RESTCONF JSON output is loaded as a Python dictionary and filtered in order to extract router identification data and separate physical interfaces from tunnel interfaces. The results were made available through a web interface, allowing the monitoring of devices from a single dashboard.

2.2. System Workflow for Data Processing and Visualization

The workflow of the proposed system is structured around an automated mechanism that collects, processes and displays the data from the network devices. For each router in the virtual network, a GET request is initiated to retrieve information about its configuration and operational status. The information is returned in a JSON-formatted file. At the data processing level, data associated with the router name and version, physical and tunnel interfaces is extracted and analysed. Each device runs the extraction and processing workflow

ensuring a distributed collecting mechanism.

The data is refreshed every 10 seconds to maintain near real-time updates for the monitored devices. In addition to configuration data, statistics are generated for each device, including the number of total interfaces, active interfaces, configured and unconfigured interfaces.

The processed information is then collected and sent to a web interface through the NGINX server where it is displayed in an organized format. NGINX was configured as a lightweight local web server responsible for hosting the dashboard interface and delivering the processed JSON-based information to the browser.

On the main tab, the Network Dashboard is displayed with two buttons: Show Architecture which illustrates the network architecture, and Router Configuration.

The Router Configuration button displays the information per device and grouped into categories such as Router Info, Physical Interfaces and Tunnel Interfaces with dedicated buttons, as illustrated in Figures 4 and 5.

Router Info section displays the device's

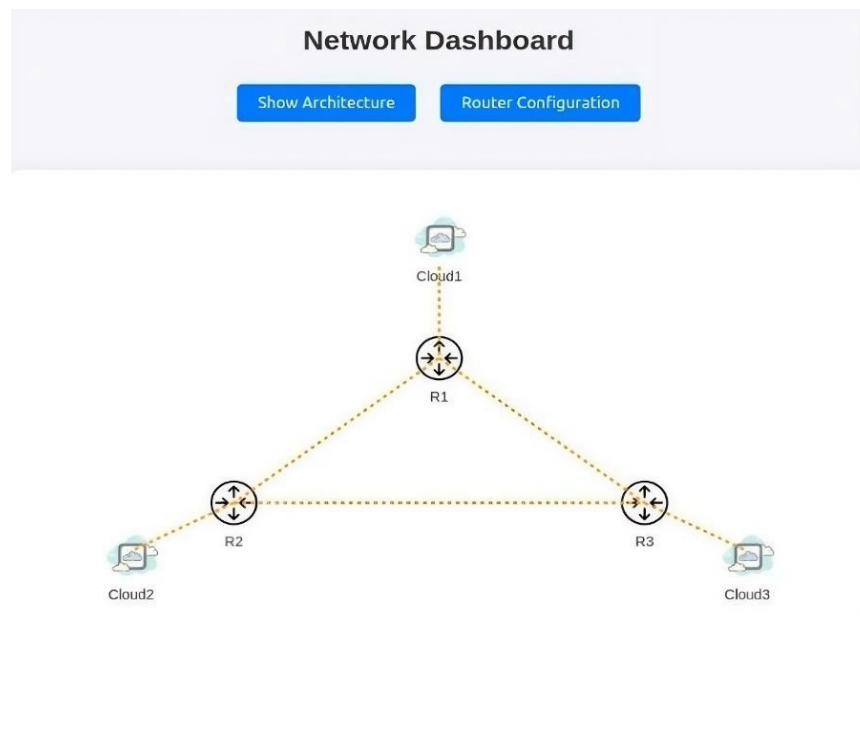


Figure 4: The Network Architecture displayed on the web page

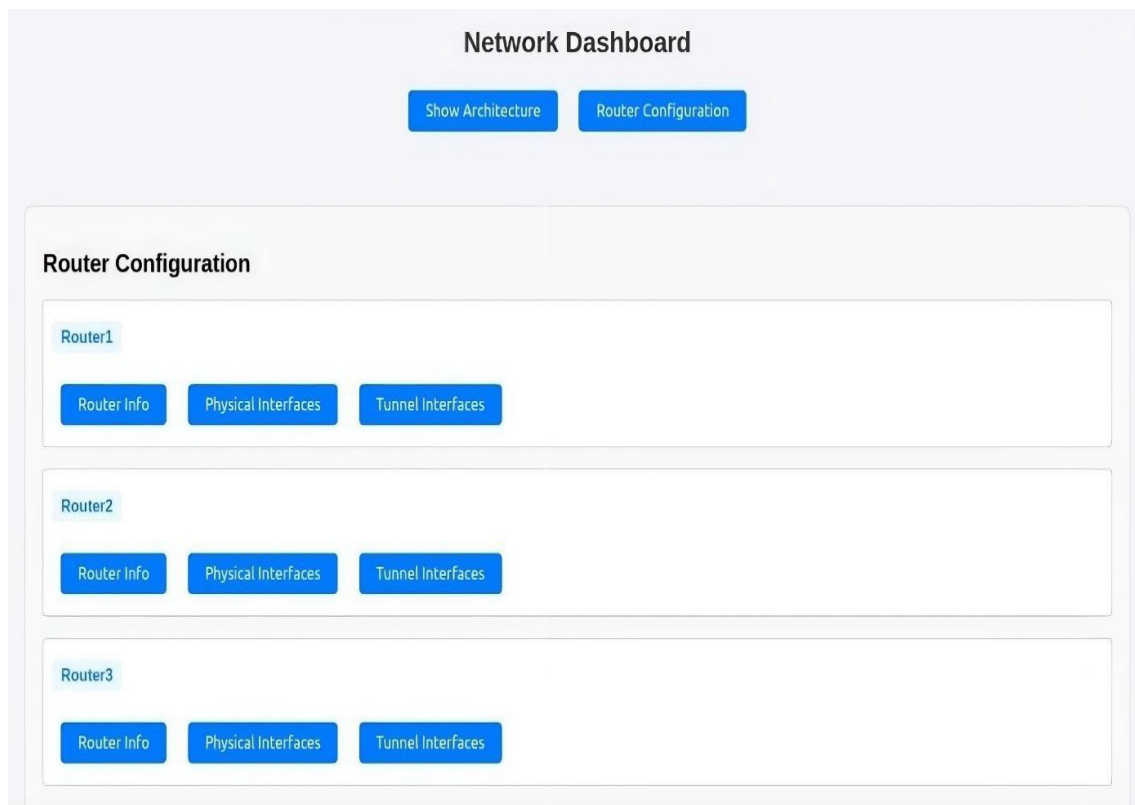


Figure 5: Organization of device sections in the web dashboard

name and version; Physical Interfaces button displays the information about physical interfaces as well as related statistics and Tunnel Interfaces button displays the equivalent information for tunnel interfaces. The described workflow ensures clear and structured data circulation between network devices and the web interface.

3. Results and Discussions

The obtained results show that the developed solution has enabled the automation and centralization of access to the configuration and operational data from devices within the network. The web interface provides a single platform for viewing all monitored devices and accessing their information without necessarily having to access the devices directly. Data is presented per device and organized into specific sections, reflecting the structure defined during the data processing stage.

This approach allows centralized visualisation of information for administrators from a single access point,

thereby eliminating the need for individual command-line sessions on each device. As a result, the time required for data verification is significantly reduced.

In addition to individual configuration details, statistics about the network and tunnel interfaces are displayed, as illustrated in Figure 6. This summarized information provides a concise overview of the device configuration state and allows faster assessment compared to manual inspection. Administrators can rapidly recognize misconfigurations of devices by analyzing the number of unconfigured or inactive interfaces.

The results highlight the efficiency of the RESTCONF-based approach for monitoring in all network environments. By relying on read-only operations, the solution is very suitable for continuous monitoring scenarios, where frequent information gathering is required without unintended alteration of configuration data.

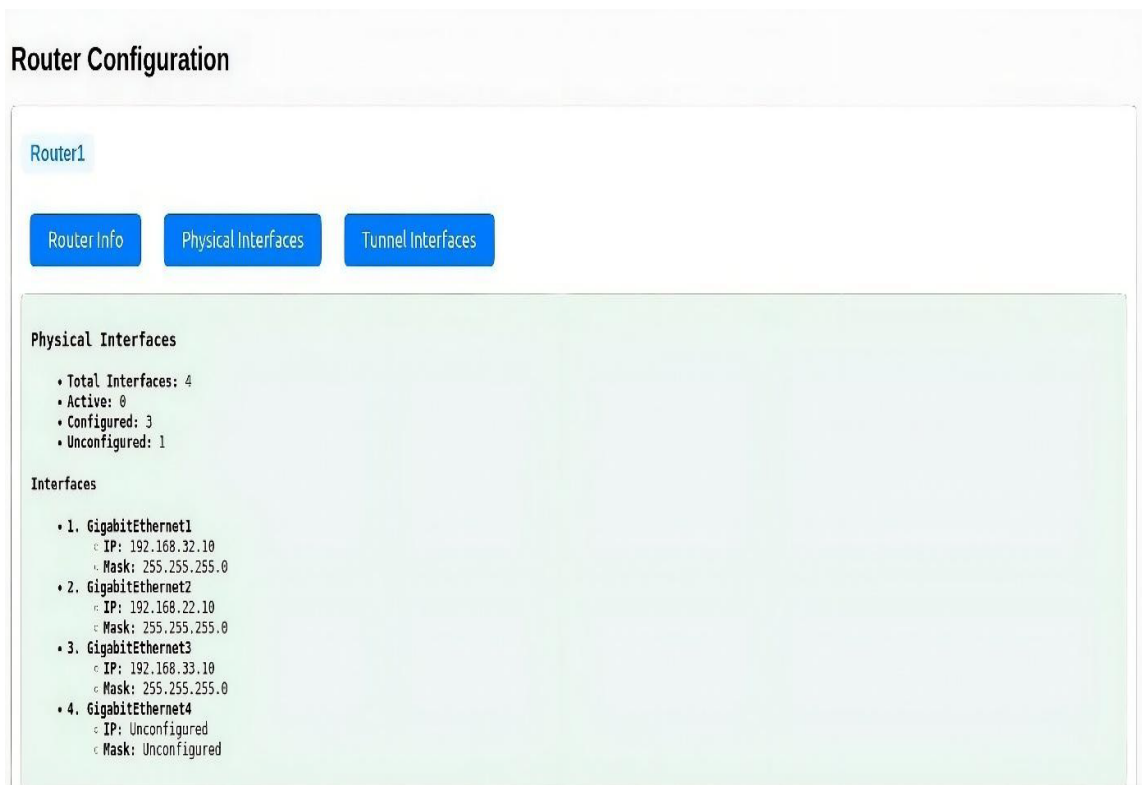


Figure 6: Physical interfaces statistics and configuration details

Although the proposed solution performs efficiently in the tested virtual environment, scalability may become a limitation in larger infrastructures. Since the system refreshes the collected data every 10 seconds, monitoring a significantly larger number of routers could introduce additional latency associated with RESTCONF GET requests and JSON processing. In large-scale environments, asynchronous or parallel request mechanisms could be considered in future implementations to improve scalability and maintain efficient monitoring performance. The experimental environment is fully virtualized and relies exclusively on Cisco CSR1000v software routers deployed in GNS3 and VMware Workstation Pro, without the use of physical network devices. The proposed approach focuses on extracting and displaying information and does not address configuration management and security solutions. This may be considered a direction for future work.

Despite these limitations, the results confirm that RESTCONF can serve as an effective

tool for data extraction and centralized monitoring in an automated manner.

In automated device management there are numerous tools already used for this purpose. Three of the most well-known tools are Oxidized, Nornir and Ansible. Each one has specific functionalities and varying levels of complexity. Oxidized is used for configuration back-ups and passive monitoring systems. Frameworks like Nornir and Ansible require advanced programming knowledge; Nornir operates through a CLI (Command Line Interface), while Ansible uses AWX, an interface complex to deploy. In contrast, the solution developed in this project provides a simple and intuitive web interface for displaying information. Designed for efficiency and ease of use, it eliminates technical barriers and simplifies network administration, making it an ideal choice for small-scale environments, where direct interaction with equipment is required. Table 1 highlights the differences between the proposed solution and several existing automation tools.

Table 1: Comparison between the proposed solution and other automation tools

Solution Criteria	Oxidized	Nornir	Ansible+AWX	Proposed Solution
Automated Configuration Extraction	Yes	Yes	Yes	Yes
Graphical Interface	Minimal	No	Full (AWX)	Lightweight dashboard
Setup Complexity	Low	High	High	Medium
Customization	Limited	High	Limited	High
Resource Consumption	Low	Medium	Medium – High	Low
Browser Accessibility	Yes	No	Through AWX	Low
Local Client Requirement	No	Yes	CLI only	No

Unlike orchestration-oriented platforms such as Ansible or framework-based solutions such as Nornir, the developed solution focuses on lightweight read-only monitoring and centralized visualization through a simplified web interface designed for rapid access to network device information.

4. Conclusion

The presented paper offers a solution for extracting and visualizing data from network devices using the RESTCONF protocol. The

proposed approach supports centralized access to device information via a web interface and minimizes the need for direct interaction with the devices.

In the future, the proposed solution may be extended to include configuration management capabilities, enhanced security controls, validation in larger-scale or physical network infrastructures and also can be customized given the open-source solutions used in the tests.

References

- [1] S. C. Pérez, H. A. Facchini, B. A. Roberti-Ferri, M. E. Stefanoni, and M. I. Césari, “Towards the Automation of Data Networks” ITECKNE. July, 2023. 20 (1). doi: 10.15332/iteckne.v20i1.2918.
- [2] T. Muhammad and M. Munir. Network Automation. European Journal of Technology. 2023. 7 (2): 23–42.
- [3] Suresh Reddy Thati, “AI-driven automation for network configuration and compliance: Transforming enterprise security posture,” World J. Adv. Res. Rev., vol. 26, no. 2, pp. 1216–1223, May 2025, doi: 10.30574/wjarr.2025.26.2.1693.
- [4] ER. SOWMITH DARAM, E. O. GOEL, and D. L. KUMAR. “Automated Network Configuration Management”. JETIR. 2026. 10 (3): i571–i587. Available: <https://www.jetir.org/view?paper=JETIR2303882>. Accessed: Jan. 07, 2026.
- [5] M. Grandérath and J. Schönwälder. A Resource Efficient Implementation of the RESTCONF Protocol for OpenWrt Systems. NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 2020, pp. 1–6, doi: 10.1109/NOMS47738.2020.9110458.
- [6] D. Mauro and K. Schmidt, Essential SNMP. “O’Reilly Media, Inc.”. 2005.
- [7] A. Bierman, M. Björklund, and K. Watsen. RESTCONF Protocol. RFC 8040. 2017. 1–137.

- [8] S. Selvaraj, “Introduction to RESTful APIs,” Mastering REST APIs. Apress. 2024. 1–18, doi: 10.1007/979-8-8688-0309-3_1.
- [9] Benoit Claise, J. Clarke, și J. Lindblad, Network Programmability with YANG. Addison-Wesley Professional. 2019.