

Network Management and Automation

Lab 10

Implementation Report

Aditi Prakash
Eric Illick
Fan Shen
Hamza Motiwalla

University of Colorado Boulder
Department of Computer Science

Index

| S.NO | TOPIC | PAGE |
|-------------|-----------------------------|-------------|
| 1. | Executive Summary | 3 |
| 2. | Components and Technologies | 4 |
| 3. | POC Network Diagram | 10 |
| 4. | POC Video Links | 11 |
| 5. | POC GitHub Repository | 11 |

Executive Summary

Respected Sir/Ma'am,

According to the discussed proposal for WDTTC Inc that we presented last week, we have implemented a proof of concept network design for the Denver DataCenter. Along with the POC network diagram, we've also developed a beta GUI to test and deploy our automation solution for the proposed network design. In this implementation report we highlight the major components for the network automation management system (NAMS). The NAMS houses the python automation code, golden configurations (core, internal and edge routers), SNMP monitor and trap python scripts, routers SSH credentials, jinja templates and GUI code which together make our network source of truth. We've currently implemented the following beta features as our proof of concept network design:

1. Active/Offline router health check
2. Fetch current router configuration details
 - a. Interface
 - b. OSPF Neighbors
 - c. BGP Neighbors
3. Fetch complete router running configuration
4. Compute router diff configuration (wrt golden config)
5. Deploy diff configuration to specific router
6. Deploy golden configurations for all routers (in parallel)
7. SNMP monitoring (CPU utilization graph)
8. SNMP trap notification (email)

We hope we've fulfilled all the necessary requirements for the network design proposal. Please feel free to contact us regarding any suggestions or questions you may have about the network design or the proof of concept implementation.

Thanks and Regards,

Cable McNetworkson
Barista Networks

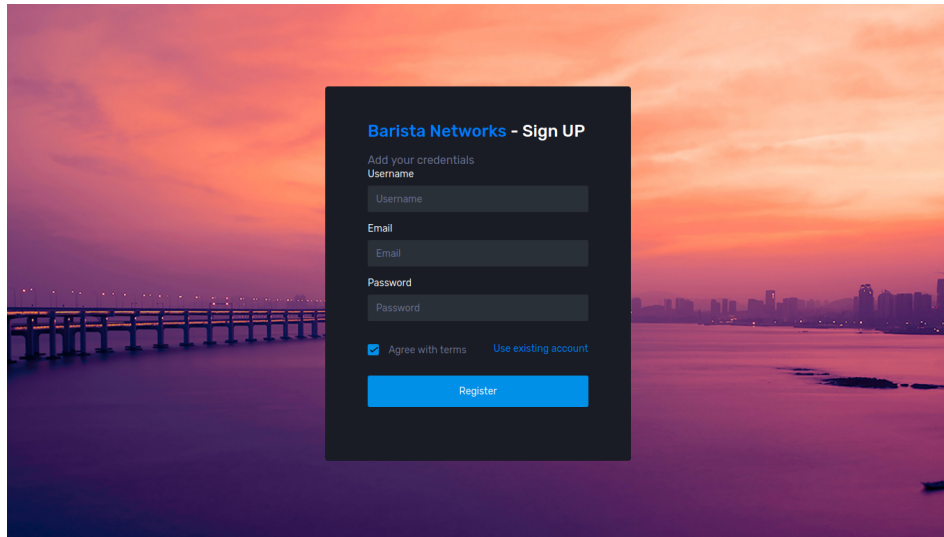
Components

Front End

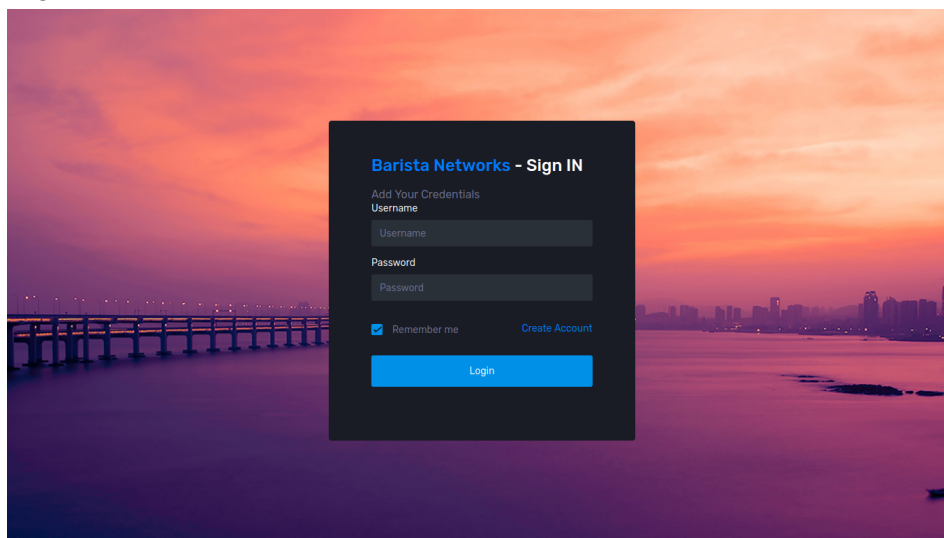
We have used HTML, CSS, Bootstrap and JQuery in the front-end.

The UI consists of the following features:

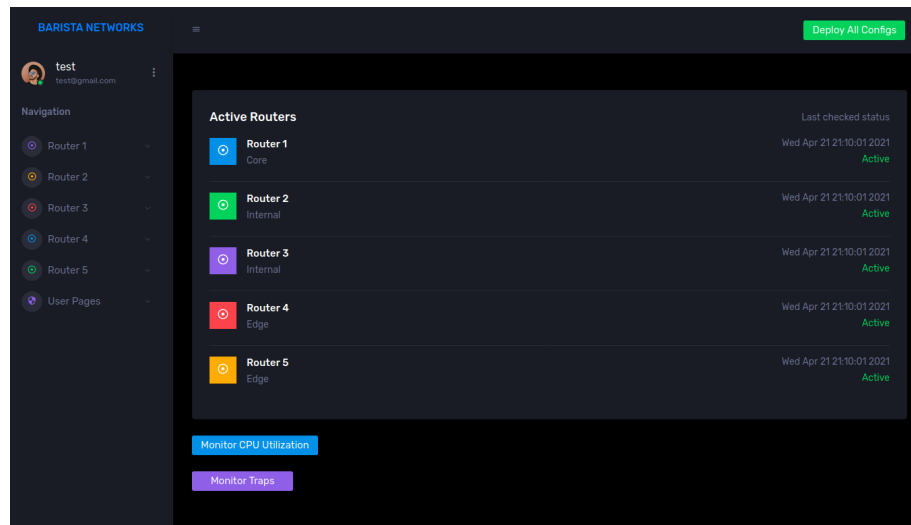
1. Sign up page
 - Requires username, email and password.

A screenshot of a web application's sign-up page. The background is a scenic image of a bridge over water at sunset. A dark, semi-transparent modal box is centered on the screen. Inside the modal, the title "Barista Networks - Sign UP" is at the top. Below it, the text "Add your credentials" is followed by the label "Username". There are three input fields: "Username", "Email", and "Password". Below the "Password" field, there is a checkbox labeled "Agree with terms" and a link "Use existing account". At the bottom of the modal is a blue button labeled "Register".

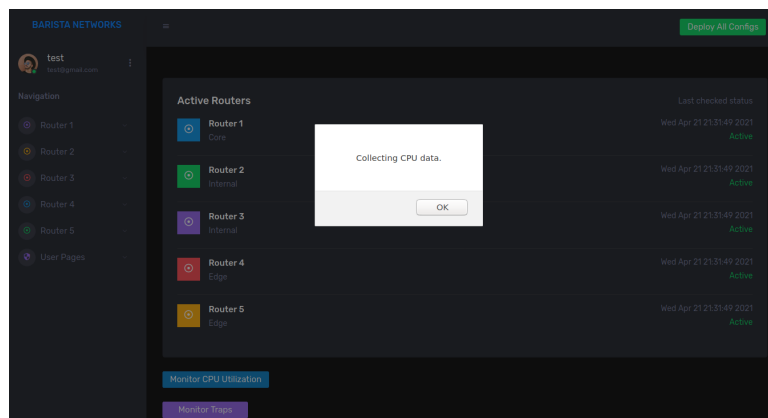
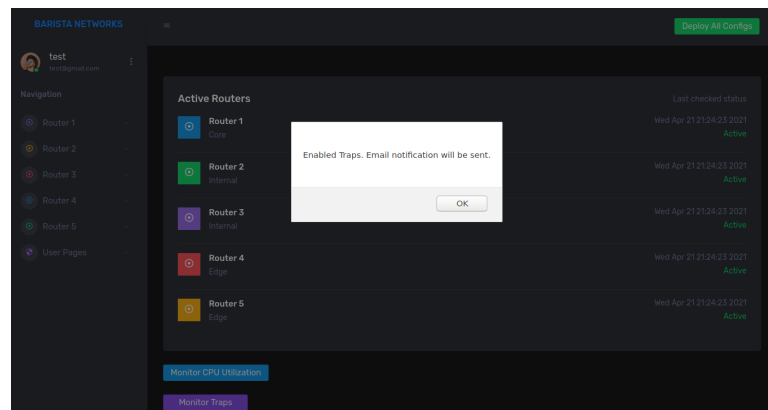
2. User login
 - Login with username / email and password.

A screenshot of a web application's login page. The background is the same scenic image of a bridge over water at sunset. A dark, semi-transparent modal box is centered on the screen. Inside the modal, the title "Barista Networks - Sign IN" is at the top. Below it, the text "Add Your Credentials" is followed by the label "Username". There are two input fields: "Username" and "Password". Below the "Password" field, there is a checkbox labeled "Remember me" and a link "Create Account". At the bottom of the modal is a blue button labeled "Login".

3. Homepage



- Navbar with:
 - Links to different router pages.
 - Button to deploy configurations to all routers.
- The dashboard displays the status of the router with the last checked timestamp.
- Buttons to enable monitoring of traps and CPU Utilization.



- The 'Deploy all Configurations' button reloads all routers with the golden configuration.

4. Router Pages

- Every router has its own page with specific configuration information.
- Config: To view current configuration in the specific router.

BARISTA NETWORKS

test

Navigation

- Router 1
- Router 2
- Router 3
- Router 4
- Router 5
- User Pages

Core_1 Configuration Details

Router Interfaces

| # | Interface | IPv4 | Prefix | Status |
|---|-----------------|---------------|--------|--------|
| 1 | FastEthernet0/0 | 198.51.100.11 | 24 | Up |
| 1 | FastEthernet1/0 | 11.1.1.1 | 30 | Up |
| 1 | FastEthernet1/1 | 10.0.0.1 | 30 | Up |
| 1 | FastEthernet2/0 | 10.0.0.21 | 30 | Up |
| 1 | Loopback0 | 11.1.1.1 | 32 | Up |

Router OSPF Neighbors

| # | Neighbor ID | Pri | State | Dead Time | Address | Interface |
|---|-------------|-----|---------|-----------|-----------|-----------------|
| 1 | 11.1.1.1 | 1 | FULL/DR | 00:00:36 | 10.0.0.22 | FastEthernet2/0 |
| 1 | 11.1.2 | 1 | FULL/DR | 00:00:33 | 10.0.0.2 | FastEthernet1/1 |

Router BGP Neighbors

| # | Local AS | Remote AS | Neighbor Address | State |
|---|----------|-----------|------------------|-------------|
| 1 | 65001 | 65001 | 11.1.1.1 | Established |
| 1 | 65001 | 1 | 11.1.2 | Established |

- Diff: To display the difference between current and golden configuration.
 - “Push Changes” button to deploy the diff config to the specific router.

BARISTA NETWORKS

Deploy All Configs

```

+username lab123 privilege 15 password lab123
+interface fa1/0
+ description Link_to_External
+ ip address 11.1.1.255.255.255.252
- no shut
+ mpls ip
+interface fa0/0
+ description Management_Port
+ ip address 198.51.100.11 255.255.255.0
- no shut
+ mpls ip
+interface fa2/0
+ description Link_to_Internal2
+ ip address 10.0.0.21 255.255.255.252
- no shut
+ mpls ip
+interface Loopback0
- no shut
+ mpls ip
+interface fa1/1
+ description Link_to_Internal1
+ ip address 10.0.0.1 255.255.255.252
- no shut
+ mpls ip
+network 10.0.0.0 0.0.0.255 area 0
+network 11.1.0 0.0.0.255 area 0
+redistribute bgp 65001
+neighbor 11.1.2 remote-as 1
+neighbor 11.1.5 remote-as 65001

```

BARISTA NETWORKS

Deploy All Configs

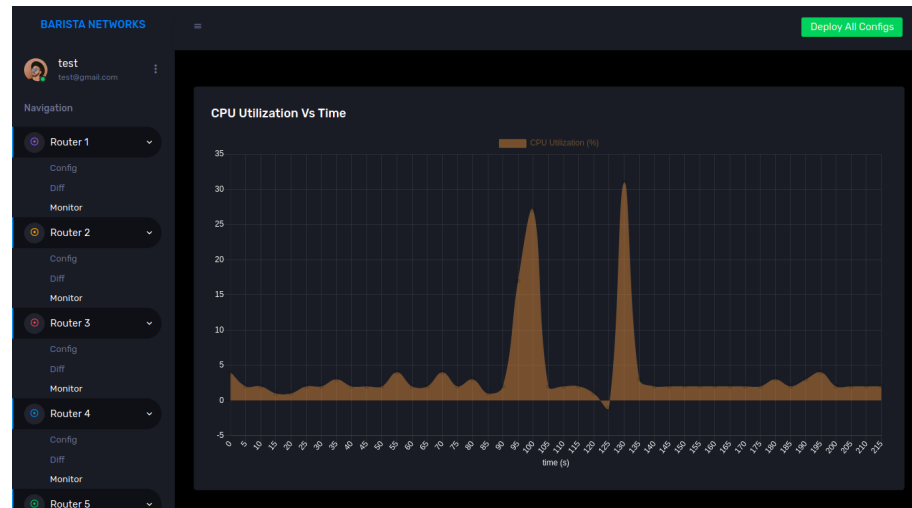
```

snmp-server enable traps vrrmib vrrf-up vrrf-down vnet-trunk-up
vnet-trunk-down
snmp-server enable traps alarms informational
snmp-server host 198.51.100.2 version 2c public
!
!
control-plane
!
line con 0
exec-timeout 0 0
privilege level 15
logging synchronous
stopbits 1
line aux 0
exec-timeout 0 0
privilege level 15
logging synchronous
stopbits 1
line vty 0 4
login local
transport input ssh
!
!
end

```

Push Differences

- **Monitor:** To monitor CPU Utilization in the specific router as a graph. The data is continually updated in the graph.



Database

For the prototype, we've used SQLite storage to store user information from the signup page. During the production stage, we plan to adopt PostgreSQL.

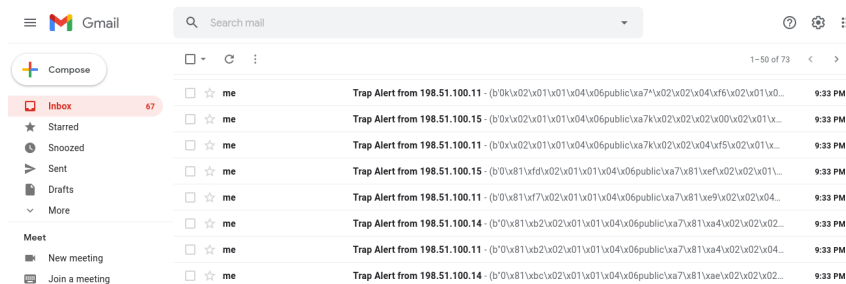
Back End

The web application uses Flask in its backend. It's design is modular with Flask blueprints. The API endpoints are defined below.

API endpoints

1. /trap

This endpoint is triggered when a user clicks on the "Monitor traps" button on the Homepage. This runs the function in the background as a separate process, independent of the web server. A python script will start running to continuously listen to UDP port 162. If there is traffic destined to UDP port 162, we know an SNMP agent is sending a trap message, an alert email with the content of that trap message will be sent to the network operator.



2. /cpu

This endpoint is triggered when a user clicks on the “Monitor CPU utilization” button on the homepage. A python script will be executed as a separate process in the thread, to collect CPU usage data from all managed devices every 5 seconds and store the data in the cpu.csv file.

3. /deploy

This endpoint is triggered when the user pushes the “Deploy all Configs” button. The endpoint calls the commit_all() python script which simultaneously loads the saved golden configuration for all the routers in the network. The commit_all() script uses the multithreading python library to deploy all the router golden configurations in parallel. This speeds up the webpage load time and response.

4. /index.html

This endpoint triggers the homepage of our GUI. Furthermore the endpoint calls the checkHealth() python script which initiates an SSH connection with all routers in parallel (multithreading). The connection is successfully established, the router status is declared as “Active”. Else if the script runs into an exception when trying to SSH into the routers, that router is given the “Offline” status.

5. /<routerName>/bgtest

This endpoint is triggered when the user pushes the “Push Changes” button on a specific router’s Diff page. The endpoint calls the commitDiff(router) python script which merges and commits the saved golden configuration with the current running configuration of the specified router.

6. /<routerName>/chartjs.html

This endpoint is triggered when the user requests to monitor a specific router. The underlying python script fetches the collected SNMP CPU utilization data and plots it as an area graph.

7. /<routerName>/basic-table.html

This endpoint is triggered when the user requests the current configuration of a specific router. The page rendered displays three different tables: router interfaces, router ospf neighbors and router bgp neighbors. Subsequently this endpoint is coupled with three underlying python script functions: getInterfaces(router), getOspfNeighbors(router) and getBgpNeighbors(router).

8. /<routerName>/typography.html

This endpoint is triggered when the user requests the “Diff” for a specific router. The endpoint calls the getConfig(router) and the getDiff(router) python scripts. getConfig(router) simply connects to the router and fetches the running configurations. getDiff(router) on the other hand, loads the saved golden configuration as a candidate configuration on the router and compares the two configurations to fetch the differences for the user. This page also provides the user with the “Push Changes” button which loads the diff configuration and commits the changes on the router configuration.

Python scripts

1. Traps

In the script “listen_to_trap.py”, we use the socket module to listen to UDP traffic destined to port 162 on the NAMS and the smtplib module to send email notifications. It is imported into “routes.py” so that it will be executed when the “Monitor traps” button is pressed.

2. CPU Usage

In the script “collect_cpu_data.py”, we use the easysnmp module to get the CPU usage data from managed devices. It is imported into “routes.py” so that it will be executed when the “Monitor CPU utilization” button is pressed.

3. Routes

All the above mentioned python scripts use the napalm library to talk to the routers. All the python scripts are contained within a single module, “routerconfig.py”. The module is imported into the “routes.py” file so that we can link each endpoint with the respective python script functionality.

Router Configuration Generator

1. Jinja2 Templates

- a. Three templates are configured to allow for Core, Internal and Edge routers to be configured.

2. Variables File

- a. The variables file is a json file that contains all the configuration information for the network.

3. Python Generation Code

- a. The python generation code uses the variables file to generate a yaml variable file for use in deploying the golden configuration files.

4. Ansible Playbook

- a. The ansible playbook uses the Jinja2 templates in combination with the yaml variables file to generate the golden configurations for each router.

POC Network Diagram

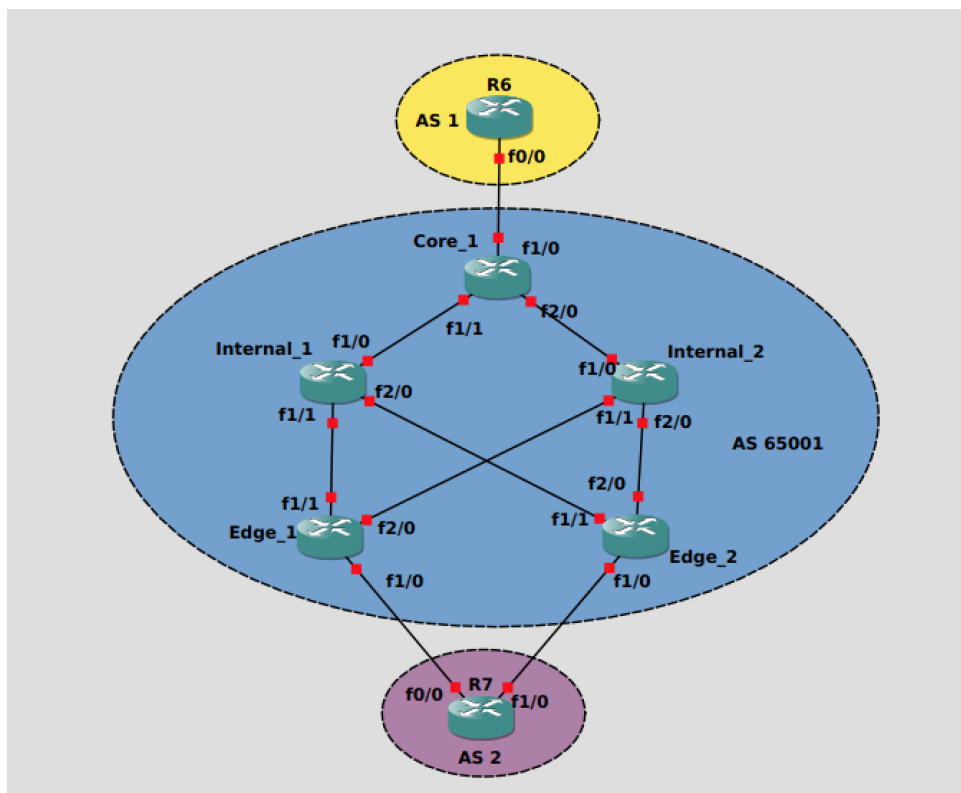


Figure 1: In-band operation network

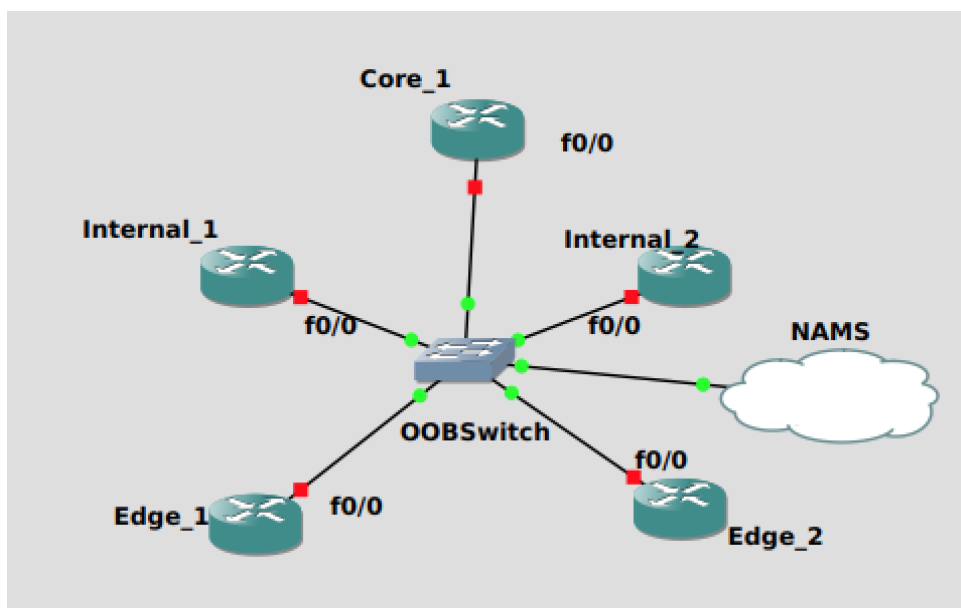


Figure 2: Out-of-band management network

POC Video Links

<https://drive.google.com/file/d/1Li-MrV7aFpJfmqwb1MmRpMIkbkv1Jh3I/view?usp=sharing>

<https://drive.google.com/file/d/1khNtXCEqUIZsq7TaMkOyx4ee17h6bzob/view?usp=sharing>

POC Github Repository

<https://github.com/aditiprakash/NetmanProject>