Software Requirements Specification (SRS)

NetOps Automation Chatbot for Cisco Devices

July 30, 2025

Contents

1 Introduction

1.1 Purpose

This document outlines the Software Requirements Specification (SRS) for the NetOps Automation Chatbot, a web-based NLP-powered system for automating network operations tasks on Cisco devices over SSH.

1.2 Intended Audience

- Network Engineers
- DevOps and NetOps Teams
- Backend and Frontend Developers
- Security Analysts
- Project Stakeholders

1.3 Scope

The chatbot allows natural language interactions from users, which are translated into executable SSH commands for Cisco devices using Netmiko and NAPALM libraries. The system uses:

- Python backend (Django)
- Frontend using Next.js hosted on Vercel
- Containerized deployment on AWS
- Firebase for JWT-based authentication
- Intent and Entity extraction with Hugging Face NLP models
- Logging, chat history, config changes in a managed database
- Queueing for config changes and real-time monitoring

2 Overall Description

2.1 Product Perspective

This chatbot acts as a bridge between non-technical users and network devices, enabling command execution, configuration changes, and querying using simple, human-readable language.

2.2 Product Functions

- NLP query parsing and intent recognition
- Secure SSH communication to Cisco routers/switches
- Persistent SSH sessions using connection pools
- Queueing of configuration changes
- Real-time monitoring of device states
- Command output parsing and user-friendly response
- Role-based access with confirmation for sensitive actions
- Slack and Microsoft Teams integration (future scope)

2.3 User Characteristics

- Network engineers who understand CLI commands
- Operators without in-depth Cisco CLI knowledge
- Admins requiring audit logs and visibility

2.4 Assumptions and Dependencies

- Devices are accessible only via SSH
- Cisco IOS-based devices only (for now)
- Internet access is available for API usage (Hugging Face, Firebase)

3 Functional Requirements

- **FR1:** User authentication via Firebase JWT
- **FR2:** Accept natural language queries
- **FR3:** Extract intent and entities from user query using Hugging Face models
- **FR4:** Establish persistent SSH sessions with Netmiko connection pooling
- **FR5:** Send commands to devices and receive output
- **FR6:** Use TextFSM to parse CLI output
- **FR7:** Queue config changes and execute sequentially
- **FR8:** Log all actions with user ID and timestamp
- **FR9:** Show confirmation prompts for sensitive changes (e.g., VLAN config, IP movement)

- **FR10:** Store chat context and logs in a persistent database
- **FR11:** Display results or summaries in frontend
- **FR12:** Auto-discover inventory of network devices

4 Non-Functional Requirements

- **NFR1:** System should respond within 3 seconds for simple queries
- **NFR2:** Should support concurrent access from 50+ users
- **NFR3:** All changes must be auditable
- **NFR4:** Communication must be encrypted (HTTPS, SSH)
- **NFR5:** Containerized backend with scalable frontend

5 Database Requirements

- Store chat context, config logs, SSH session metadata
- Use a managed DB (PostgreSQL, MongoDB, or Firebase Firestore)
- Schema includes:
  - `users: {userId, role, authToken}`
  - `logs: {timestamp, userId, action, device, configBlock}`
  - `chatHistory: {userId, sessionId, message, timestamp}`

6 System Architecture

6.1 Components

- Frontend: Next.js app with chat UI, token-based auth
- Backend: Django REST API to handle user queries, execute SSH commands, return results
- SSH Daemon: Persistent pool of SSH connections
- NLP Engine: Hugging Face Transformers for intent/entity detection
- Database: Managed service (e.g., Firebase, MongoDB Atlas, or PostgreSQL on RDS)

6.2 Data Flow

1. User sends a query in natural language
2. Query is parsed and intent/entities extracted
3. Command is formulated and checked for sensitivity
4. If sensitive, user is prompted for confirmation
5. Else, command is queued and executed via SSH pool
6. Output is parsed with TextFSM and shown to user
7. Logs are written with context and metadata

7 NLP + Output Customization

7.1 Sample Intent and Entity Extractor

User Query: "Show me the IP route table of Switch-Core-1"
Extracted Intent: `show_command`
Entity:

- ‘device$_name$‘ : $Switch - Core - 1$‘$command$‘ : $showiproute$

7.2 Customizing CLI Output

- Use TextFSM templates to convert raw CLI output into structured data (JSON/table).
- Use the user's original question to decide how much to summarize (e.g., "just active routes").
- Optionally route the parsed CLI output to an LLM (OpenAI, Gemini) to provide summarization if complexity demands.

7.3 TextFSM Example

Template for ‘show ip interface brief‘:

```
Value Required INTERFACE (\S+)
Value IP_ADDRESS (\S+)
Value OK (\S+)
Value METHOD (\S+)
Value STATUS (up|down|administratively down)
Value PROTOCOL (up|down)

Start
  ^${INTERFACE}\s+${IP_ADDRESS}\s+${OK}\s+${METHOD}\s+${STATUS}\s+${PROTOCOL} -> Reco
```

8 Future Enhancements

- Multi-vendor support (Juniper, Arista)
- Real-time alerting on network faults
- Self-healing actions
- Slack/MS Teams integration

9 Appendix

9.1 References

- Netmiko GitHub
- NAPALM GitHub
- TextFSM Docs
- Hugging Face Models