Network Programmability and Automation

1 Network Programmability and Automation

1.1 Introduction to Network Programmability and Automation

1.1.1 Typical O&M Scenarios

1.1.1.1 Device Upgrade

- Perform periodic batch upgrades on thousands of live network devices.

1.1.1.2 Configuration Audit

- Annual audits to ensure compliance (e.g., STelnet enabled, spanning tree security configured).

1.1.1.3 Configuration Change

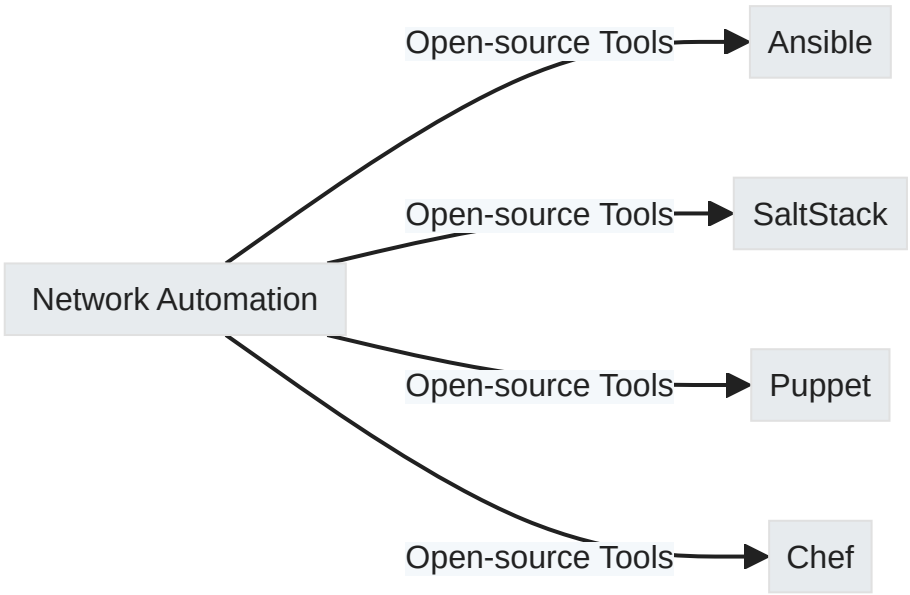
- Regularly update device accounts and passwords due to security requirements.

1.1.2 Network Automation

1.1.2.1 Introduction

> Network automation reduces human dependency by using tools for automated deployment, operations and O&M.

1.1.2.2 Tools for Automation



- Recommended to acquire **coding skills** for network engineering capability construction.

1.1.3 Programming-based Network Automation

1.1.3.1 Python as a Skill Requirement

- Python is essential for writing automation scripts that handle repetitive and rule-based tasks.

1.1.3.2 Automated Device Configuration Example

---

1. **Write Configuration File**

```yaml
Sysname SW1
Vlan 10 description A
Vlan 20 description B
VLAN 30 description C
```

2. **Push Configuration with Python**
   - Establish SSH/Telnet connection.
   - Upload configuration script to the network device.

1.1.3.3 Python Scripting Basics

```python
# Python pseudo-code example for pushing configuration:
import some_ssh_library

def push_config(device_ip, config_file):
    # Establish SSH connection to the device using the library's functions.
    # Open the config file and read the commands.
```

```
7       # Iterate over commands and execute them on the
    device via SSH.
8       pass  # Implementation details here.
```

1.1.3.4 Benefits of Network Automation

> With network automation, configurations can be deployed more efficiently and consistently across multiple devices.

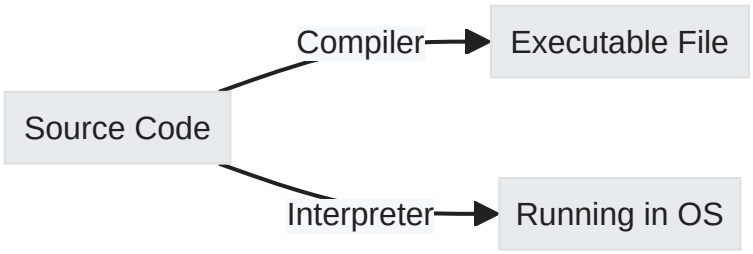
1.2 Overview of Programming Language and Python

1.2.1 Overview

- Programming languages control computer behavior.
- Classified into **compiled** and **interpreted** languages.
- **Based on language levels:** machine language, assembly language, high-level language.

1.2.2 Compiled vs. Interpreted

- **Compiled:** Translated into machine code before execution (e.g., C/C++).
- **Interpreted:** Translated at runtime, line by line (e.g., Python).



1.2.3 Language Level Classification

1.2.3.1 Machine Language

- Consists of `0` and `1` instructions.

1.2.3.2 Assembly Language

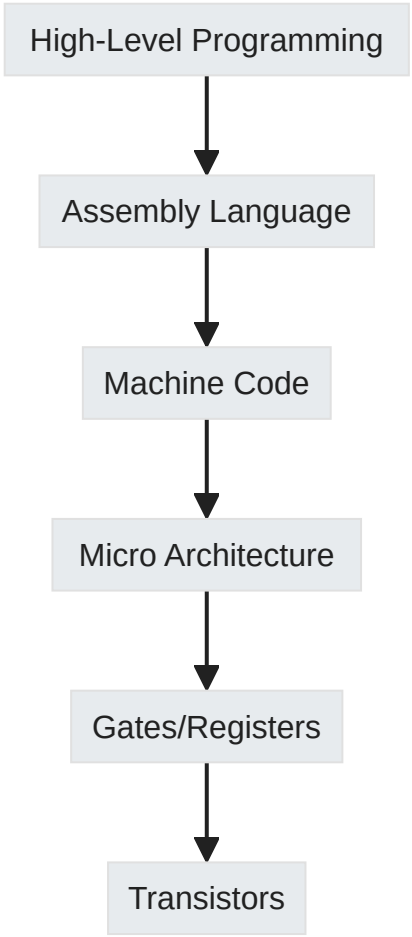
- Encapsulates hardware instructions for readability (e.g., `MOV`, `ADD`).

1.2.3.3 High-Level Language

- Includes languages like Java, Python.
- Requires translation to machine code.

> - **JVM:** Java virtual machine
> - **PVM:** Python VM

1.2.4 Computing Stack & Program Execution



1.2.5 Python Overview

> Python is a high-level interpreted language created by Guido Van Rossum.

1.2.5.1 Advantages & Disadvantages of Python

Advantages & Disadvantages

- **Advantages:**
  - Python is written clearly and is easy to read.
  - It can be used in different ways, both simple and complex.
  - There are lots of extra tools you can get to do more things.

- **Disadvantages:**
  - It can run slowly because it processes one instruction at a time.

1.2.6 Python Execution Process

---

1. Install the Python environment on the operating system.
2. Write the source code `.py`.
3. Compiler generates bytecode `.pyc`.
4. Python VM executes bytecode.

```python
# Example Code Block:
temp = v[k]
v[k] = v[k+1]
v[k+1] = temp
```

1.2.7 Getting Started with Python

1.2.7.1 Interactive Mode

Enter commands directly into the interpreter:

```python
>>> print("hello world")
hello world
>>> a = 1; b = 2; print(a + b)
3
```

1.2.7.2 Script Mode

Write scripts and execute them:

```shell
$ python demo.py # Executes script demo.py containing python code.
```

demo.py:

```python
print("hello world")
a = 1; b = 2; print(a + b)
```

1.2.8 Code Style Guide for Python

1.2.8.1 Indentation & Comments

- Use four spaces per indentation level.
- Single-line comments start with `#`.
- Multi-line comments use triple quotes `"""`.

1.2.8.2 Identifier Naming Rules

Identifiers should start with a letter or underscore:

Correct	Incorrect
user_id	4user_id

1.2.9 Python Functions & Modules

Functions improve modularity; modules contain reusable definitions:

Function Example:

```python
def greet():
    print("Hello World")
greet() # Outputs "Hello World"
```

Module Example:

demo.py:

```python
def sit():
    print('A dog is now sitting')
```

test.py:

```python
import demo
demo.sit() # Calls function from demo module.
```

1.2.10 Introduction to telnetlib

The `telnetlib` module in Python allows you to work with Telnet, a protocol used to establish a command-line interface connection to remote computers over the internet or local networks.

```python
from telnetlib import Telnet
tn = Telnet(host=None, port=0[, timeout])
tn.read_all()
```

Methods:

- `read_until(expected, timeout=None)` : Keep reading data until you find a specific text or until a certain amount of time has passed.
- `read_all()` : Keep reading data until there is no more coming (End Of File - EOF).
- `read_very_eager()` : Read any available data right away without waiting, but don't wait around for more data if there isn't any immediately available.
- `write(buffer)` : Send your own commands or data to the server.
- `close()` : End your Telnet session and close the connection.

1.3 Cases

1.3.1 Telnet Login Using telnetlib

1.3.1.1 Overview

- **Objective**: Log into a network device functioning as a Telnet server using Python's `telnetlib` .

1.3.1.2 Manual Verification

> Use terminal Telnet client for manual login verification.

1. Run command: `telnet 192.168.10.10`
2. Enter password when prompted.

Shell

```
C:\Users\Richard>telnet 192.168.10.10
```

```
Password:
Info: The max number of VTY users is 5...
<Huawei>
```

1.3.1.3 Python Code Example

Python

```python
import telnetlib

host = '192.168.10.10'
password = 'Huawei@123'

tn = telnetlib.Telnet(host)
tn.read_until(b'Password:')
tn.write(password.encode('ascii') + b"\n")

print(tn.read_until(b'<Huawei>').decode('ascii'))
tn.close()
```

> In Python, use `.encode()` to convert strings to ASCII format required by telentlib; prefix byte strings with `b` (e.g., b'Password:') to denote bytes objects as per official requirements.

1.3.1.4 Code Explanation

- Importing module.
- Setting host IP and password.
- Creating a Telnet connection.
- Reading output until 'Password:' prompt.
- Sending encoded password followed by newline.
- Reading output until `<Huawei>` prompt appears.

- Closing the connection.

1.3.1.5 Mermaid Flowchart