Home

My name is Ganesh Venepally. I'm a Network Engineer with 15+ years of experience. Most part of my career I worked for major financial banks building and supporting thier complex networks. My technical background includes network design,implementations and operations.

Here I try to write about interesting technical challenges I face in my daily work and I hope you'll find something helpful to take away.

If you want to get in touch you can PM me on twitter.

Network Automation

Welcome to my blog on Network Automation! In today's fast-paced business environment, organizations are constantly looking for ways to improve the efficiency and reliability of their network operations. One of the key ways to achieve this is through automation.

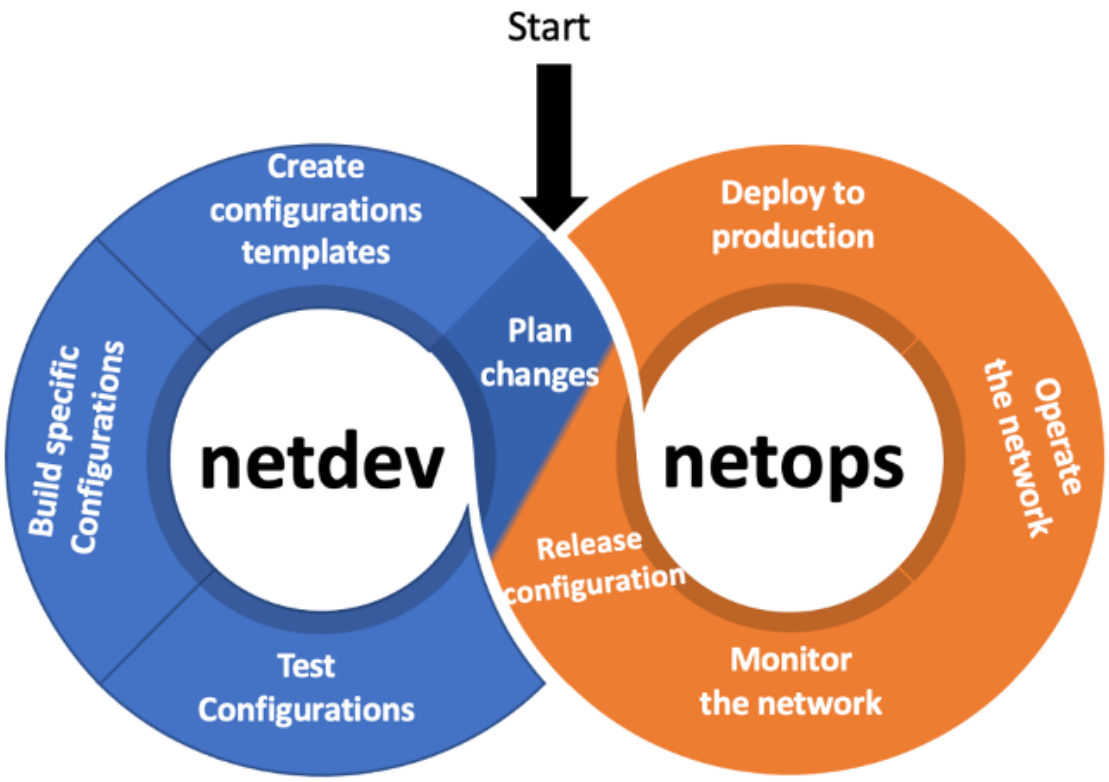
Network automation is the use of software and tools to automate repetitive or complex tasks associated with the management and configuration of network devices. This can include tasks such as configuring and provisioning new devices, monitoring and troubleshooting network issues, and managing software and security updates.

The goal of network automation is to increase the speed and reliability of network operations, while reducing the risk of errors and downtime. With automation, network engineers can focus on more strategic and value-added tasks, such as designing and implementing new network solutions, rather than spending their time on manual and repetitive tasks.

In this blog, I will explore various aspects of network automation, including the use of automation tools, best practices for implementing automation, and how automation can be used to improve the security and performance of networks.

Whether you are new to network automation or an experienced network engineer looking to expand your skills, this blog is designed to provide valuable insights and tips to help you improve the efficiency and effectiveness of your network operations.

NetDevOps



NetDevOps Pipeline

DevOps is a set of practices and tools that aim to improve collaboration and communication between development and operations teams in order to increase the speed and quality of software delivery. In the context of network operations, DevOps practices can be applied to the management and automation of network infrastructure in order to improve the speed and reliability of network changes, as well as reduce the risk of errors and downtime.

One key aspect of DevOps in network context is the use of automation and scripting to manage network devices and configurations. This can include using tools such as Ansible, Puppet, or Chef to automate the configuration of network devices, as well as using version control systems such as Git to manage and track changes to network configurations.

CI/CD - Another important aspect of DevOps in network context is the use of continuous integration and continuous delivery (CI/CD) to manage the deployment and testing of network changes. This can include using automated testing and validation tools to ensure that changes to the network are safe and functional before they are deployed to production. Additionally, it also includes monitoring and metrics tools to track the performance and availability of the network.

Data Models and Encodings

Data models and encodings are important concepts in networking that are used to represent and manage the configuration and state of network devices and services.

A data model is a structured representation of the data that a network device or service uses to configure and operate. This can include information such as network topology, device configurations, and performance metrics. Data models are typically defined using formal languages such as YANG, which is used to model network configurations and state data for various protocols such as IP, MPLS, and LACP.

An encoding is a method used to represent the data defined by a data model in a format that can be transmitted and processed by network devices and management systems. There are several encoding formats that are commonly used in networking, including XML, JSON. Each encoding format has its own strengths and weaknesses, and the choice of encoding format will depend on the specific requirements of the network and the devices and systems that will be using the data.

In Networking, data models are used to define the structure of the data, while encodings are used to represent the data in a format that can be transmitted over the network. For example, a YANG model can be used to define the structure of the data for a particular protocol such as BGP, and then that data can be encoded in JSON format to be transmitted over the network and processed by the devices and systems that are using the BGP protocol.

One of the main benefits of using data models and encodings in networking is that they allow for a more consistent and efficient management of network devices and services. Data models provide a standard way to represent the data, which allows for better interoperability between devices and systems.

YANG

YANG (Yet Another Next Generation) is a data modeling language used to model the configuration and state data for various network protocols such as IP, MPLS, and LACP. It is used by several networking vendors, such as Cisco and Juniper, to define the data models for their devices and services. For example, a YANG data model can be used to define the structure of the data for a particular protocol such as BGP, and then that data can be encoded in JSON format to be transmitted over the network and processed by the devices and systems that are using the BGP protocol.

Index

Python is a powerful programming language that has become increasingly popular in the field of network automation. With its simple syntax, vast libraries, and flexibility, Python makes it easy to automate tasks such as network configuration, monitoring, and troubleshooting.

This Section is dedicated to providing tips, tutorials, and best practices for using Python to automate network tasks and improve network management. It will cover topics such as using Python libraries such as Netmiko, NAPALM, and PyEZ to automate network device configuration and management, using Python to perform network monitoring and troubleshooting, and using Python to integrate with network automation tools such as Ansible, and Nornir.

Virtual Environment

Creating a Python virtual environment is a good practice to isolate your Python project's dependencies and avoid conflicts with other projects. Here's an example of how you can create a virtual environment using the venv module in Python 3:

Example

- Open a command prompt and navigate to the directory where you want to create the virtual environment.
- Run the following command to create a new virtual environment:

```
python -m venv myenv
```

This will create a new directory called "myenv" that contains the files needed for the virtual environment.

- To activate the virtual environment, run the following command:

```
myenv\Scripts\activate.bat
```

- *Once the virtual environment is activated, you should see the name of the virtual environment in the command prompt, for example:

```
(myenv) C:\myproject>
```

- To install packages in the virtual environment, you can use pip, for example:

```
(myenv) C:\myproject> pip install requests
```

- To deactivate the virtual environment, you can simply run the command:

```
deactivate
```

> It's worth noting that the myenv directory is a copy of the python executable and all the standard libraries, this means that the virtual environment is not dependent of the global python installation and you can have multiple versions of a package or libraries installed.

Paramiko

Paramiko is a Python library that provides an SSH2 protocol implementation for secure and automated access to network devices such as routers and switches. It allows you to establish an SSH connection to a network device and execute commands or transfer files.

Examples

Here are a few examples of how you can use paramiko to automate tasks on Cisco routers:

- Connecting to a Cisco router: Paramiko can be used to establish an SSH connection to a Cisco router. Here's an example of how to connect to a router using Paramiko:

```
import paramiko

# Create a new SSH client
client = paramiko.SSHClient()

# Add the Cisco router's IP address and username/password to connect
client.connect('192.168.1.1', username='admin', password='password')
```

- Running commands on a Cisco router: Once connected, Paramiko can be used to run commands on a Cisco router. Here's an example of how to run the "show version" command on a router:

```
# Open an SSH channel
channel = client.invoke_shell()

# Send the command to the router
channel.send('show version\n')

# Receive the output from the router
output = channel.recv(9999)
print(output.decode())
```

- Transferring files to/from a Cisco router: Paramiko can also be used to transfer files to and from a Cisco router. Here's an example of how to transfer a file to a Cisco router:

```
# Open an SFTP session
sftp = client.open_sftp()

# Transfer the file to the router
sftp.put('local_file.txt', 'remote_file.txt')
```

Netmiko

Netmiko is a Python library built on top of Paramiko, which simplifies automating network device configuration and management tasks. It is designed to work with Cisco, Arista, Juniper, and other network vendors.

Examples

Here are a few examples of how Netmiko can be used to interact with Cisco routers:

- Connecting to a Cisco router: Netmiko can be used to establish an SSH connection to a Cisco router. Here's an example of how to connect to a router using Netmiko:

```
from netmiko import ConnectHandler

# Define the device parameters
device = {
    'device_type': 'cisco_ios',
    'ip': '192.168.1.1',
    'username': 'admin',
    'password': 'password',
}

# Connect to the device
net_connect = ConnectHandler(**device)
```

- Running commands on a Cisco router: Once connected, Netmiko can be used to run commands on a Cisco router. Here's an example of how to run the "show version" command on a router:

```
output = net_connect.send_command("show version")
print(output)
```

- Configuring a Cisco router: Netmiko can also be used to configure Cisco routers. Here's an example of how to configure an interface on a Cisco router:

```
# Define the configuration commands
commands = [
    'interface FastEthernet0/1',
    'description Configured by Netmiko',
    'ip address 192.168.1.10 255.255.255.0',
    'no shutdown'
]

# Send the configuration commands to the device
net_connect.send_config_set(commands)
```

Nornir

Nornir is a Python automation framework that is designed to automate network tasks, it's built on top of the popular libraries such as Netmiko and Napalm. It allows for parallel execution of tasks, and provides an easy-to-use inventory system to manage devices.

Examples

Here are a few examples of how Nornir can be used to interact with Cisco routers:

- Creating an inventory: Nornir uses an inventory system to manage the devices. Here's an example of how to create an inventory of Cisco routers:

```
from nornir import InitNornir
from nornir.plugins.inventory import SimpleInventory

# Define the devices
devices = {
    "cisco_router1": {
        "hostname": "192.168.1.1",
        "username": "admin",
        "password": "password",
        "platform": "cisco_ios",
    },
    "cisco_router2": {
        "hostname": "192.168.1.2",
        "username": "admin",
        "password": "password",
        "platform": "cisco_ios",
    },
}

# Create the inventory
nr = InitNornir(inventory=SimpleInventory(hosts=devices))
```

- Running commands on a Cisco router: Once connected, Nornir can be used to run commands on a Cisco router. Here's an example of how to run the "show version" command on a router using Nornir:

```
from nornir import InitNornir
from nornir.plugins.tasks import networking

# Initialize Nornir
nr = InitNornir(inventory={'options': {'host_file': 'hosts.yaml'}})

# Run the command on the device
result = nr.run(networking.netmiko_send_command, command_string='show
version')
```

Data Models and Encodings

Understanding how data can be structured and encoded is very important in programming in general and network automation in particular.

YANG & Openconfig

YANG (Yet Another Next Generation) is a data modeling language originally developed for NETCONF and defined in RFC 6020 and then updated in RFC 7950. YANG and NETCONF can be considered as successors to SMIng and SNMP respectively.

> YANG provides a format-independent way to describe a data model that can be represented in XML or JSON.
>
> *Jason Edelman, Scott S. Lowe, Matt Oswalt. Network Programmability and Automation, p. 183*

There are hundreds of YANG data models available both vendor-neutral and vendor-specific. The YANG catalog web site can be helpful if you need to find data models relevant to your tasks.

Because of this abundance of data models and lack of coordination between standards developing organizations and vendors it seems that YANG and NETCONF are going the same path SNMP went (i.e. used only for data retrieval, but not configuration). OpenConfig workgroup tries to solve this by providing vendor-neutral data models, but I think that Ivan Pepelnjak's point from 2018 stating that "*seamless multi-vendor network device configuration is still a pipe dream*" still holds in 2020.

XML

XML (eXtensible Markup Language) although a bit old is still widely used in various APIs. It uses tags to encode data hence is a bit hard to read by humans. It was initially designed for documents but is suitable to represent arbitrary data structures.

You can refer to this tutorial to learn more about XML.

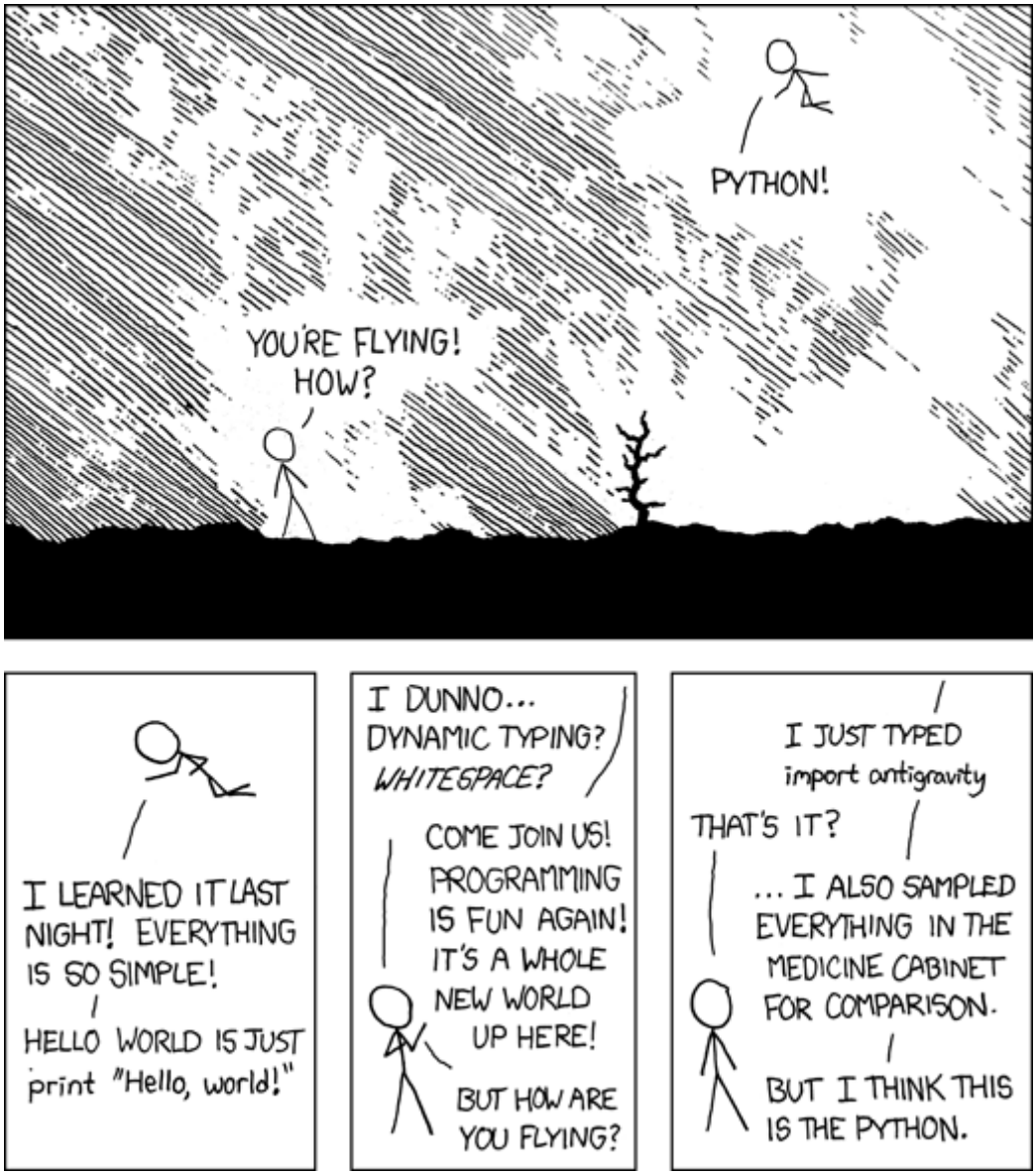
Let's see how this sample CLI output of Cisco IOS `show vlan` command can be encoded with XML:

Technologies

This section is quite opinionated and aims to introduce you to the essential tools leaving behind many others for the sake of brevity. I highly recommend to take a look at the Awesome Network Automation list later.

Python

Python is a go-to programming language when it comes to network automation. All of the popular network automation tools and libraries are written in Python.



Python

Due to its gentle learning curve and immense popularity (second most used language on GitHub after JavaScript as of the time of writing), Python is a great choice to get started with programming.

Interacting with network devices

There are two major ways of accessing network devices programmatically: CLI and API.

CLI

For a long time, the only API of network devices was CLI which is designed to be used by humans and not automation scripts. These are the main drawbacks of using CLI as an API: * **Inconsistent output**
The same command outputs may differ from one NOS (Network Operating System) version to another. * **Unstructured data**
Data returned by command execution in CLI is plain text, which means you have to manually parse it (i.e. CLI scraping) * **Unreliable command execution**
You don't get a status code of an executed command and have to parse the output to determine whether the command succeeded or failed.

Despite more and more networking vendors begin to include API support in their products it's unlikely that you won't have to deal with CLI during your network automation journey.

To parse CLI output regular expressions are used. Not a very user-friendly technology to put it mildly.

> "I don't know who you are. I don't know what you want. If you are looking for technical help, I can tell you I don't have any time. But what I do have are a very particular set of regexes. Regexes I have acquired over a very long career. Regexes that are a nightmare for people like you to debug. If you leave me alone now, that'll be the end of it. I will not look for you, I will not pursue you, but if you don't, I will look for you, I will find you and I will use them in your code."
>
> Quotes from the Cloudiest WebScaliest DevOps Teams

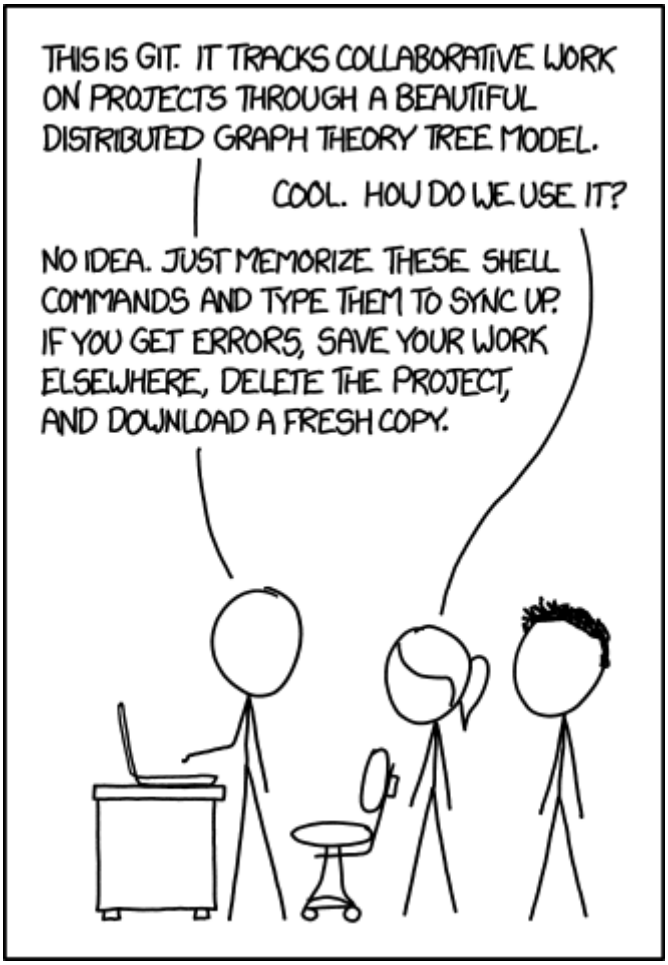
Fortunately, there are a lot of tools and libraries today that make CLI scraping easier by doing a lot of the regex heavy lifting.

APIs

If you are lucky and devices in your network have an API or maybe are even driven by SDN controller this section is for you. Network APIs fall into two major categories: HTTP-based and NETCONF-based.

Git

This section covers basic Git usage and terminology. But first, I'd like to highlight several reasons why you should care about Git and version control in the first place.



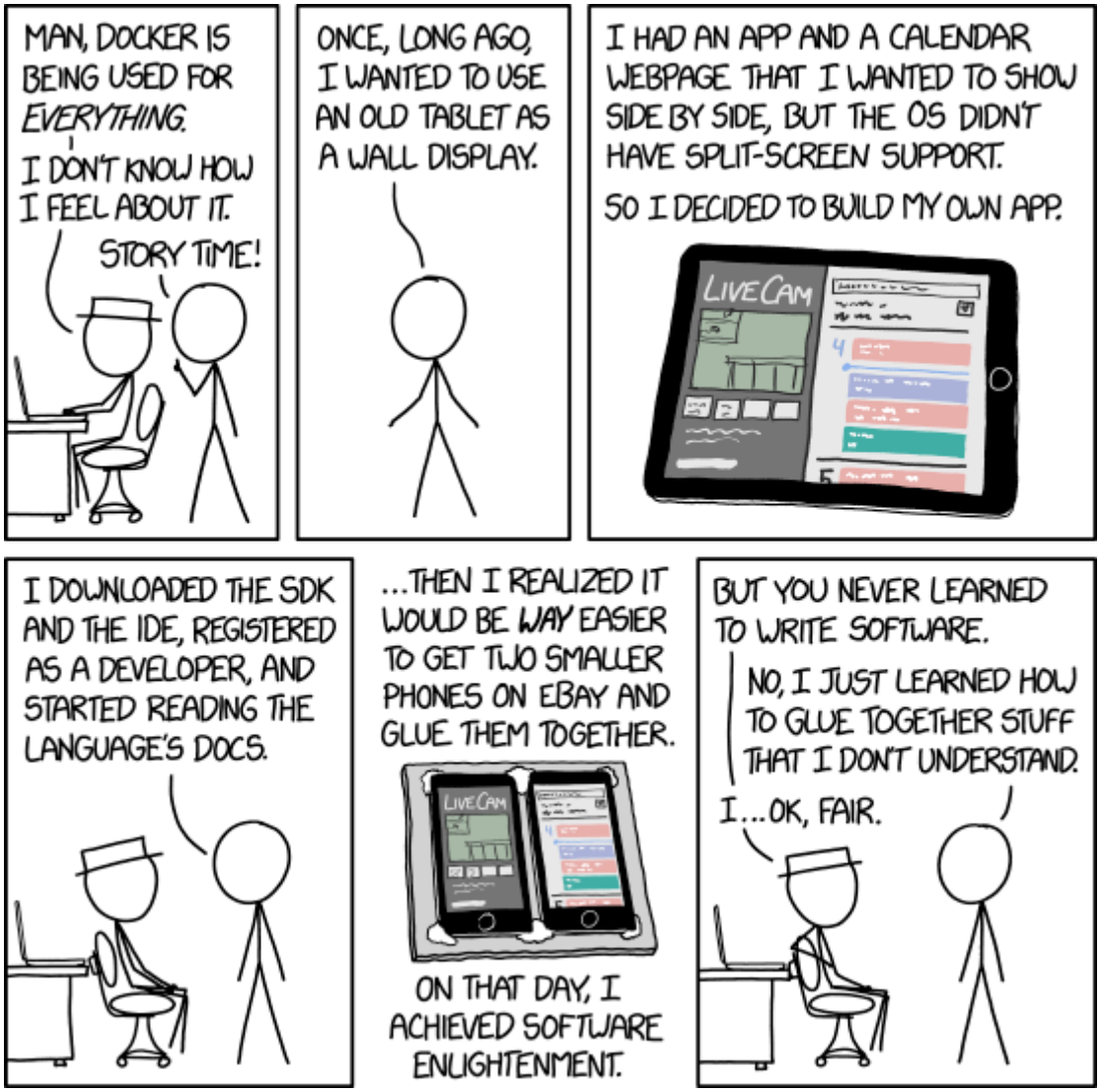
Git

Why use Git?

- **Visibility & control**
  By placing your scripts, configuration templates, or even device configurations in Git you can start tracking all the changes and rollback to previous versions if needed.
- **Experimenting**
  When working on a new feature it's very convenient to create a new branch in the same Git repository rather than copy the whole working directory to a new place.

Docker and containers

Linux containers have been around for quite a long time (and chroot and jail even longer) but Docker was what made it popular and accessible.



Containers

Containers allow to run software in an isolated environment, but contrary to VMs each container doesn't need a full-blown OS to run. This makes containers more resource-effective in terms of CPU, RAM, and storage not to mention that you don't need to maintain a separate OS for each container as with VMs.

Automation Tools

Here I would like to give you a quick overview of the most popular and prominent network automation tools.

Connection Management and CLI Scraping

Netmiko

Netmiko is a Python library based on paramiko and aimed to simplify SSH access to network devices. Created by Kirk Byers in 2014 this Python library stays the most popular and widely used tool for managing SSH connections to network devices.

Scrapli

Scrapli is a somewhat new python library (first release in 2019) that solves the same problems as Netmiko but aims to be "*as fast and flexible as possible*".

Parsing

TextFSM and NTC Templates

TextFSM is a Python module created by Google which purpose is to parse semi-formatted text (i.e. CLI output). It takes a template file and text as input and produces structured output. NTC templates is a collection of TextFSM templates for a variety of networking vendors. TextFSM can be used in conjunction with netmiko and scrapli.

TTP (Template Text Parser)

TTP is the newest addition to the text parsing tools. It's also based on templates that resemble Jinja2 syntax but work in reverse. A simple TTP template looks much like the text it is aimed to parse but the parts you want to extract are put in {{ curly braces }}. It doesn't have a collection of prebuilt templates but given its relative ease of use, you can quickly create your own.

PyATS & Genie

These internal Cisco tools were publicly released a few years back and continue to develop rapidly. PyATS is a testing and automation framework. It has a lot to it and I encourage you to learn about it on Cisco DevNet resources. Here I would like to focus on two libraries within the PyATS framework: Genie parser and Dq. The first one as the name implies is aimed to parse CLI output and has a huge collection (2000+) of ready-made parsers for various devices (not limited to Cisco). The second one, Dq, is a great time saver when you need to access the parsed data. Often parsers such as Genie return data in complex data structures (e.g. nested dictionaries) and to access something you would need loops if statements and a strong understanding of where to look. With Dq, you can make queries without much caring of where in a nested structure your data resides.

Configuring devices

NAPALM

As the official documentation states

> NAPALM (Network Automation and Programmability Abstraction Layer with Multivendor support) is a Python library that implements a set of functions to interact with different network device Operating Systems using a unified API.

Supported devices

As of the time of writing NAPALM supported the following network operating systems:

- Arista EOS
- Cisco IOS
- Cisco IOS-XR
- Cisco NX-OS
- Juniper JunOS

Working with device configuration

With NAPALM you can push configuration and retrieve operational data from devices. When manipulating device configuration you have two options:

- **Replace** the entire running configuration with a new one
- **Merge** the existing running configuration with a new one

Replace and merge operations don't apply at once. Before committing the new configuration you can compare it to the currently running configuration and then either commit or discard it. And even after applying the new config, you have an option to rollback to the previously committed configuration if the network OS supports it.

Validating deployment

The ability to retrieve operational data from devices brings a powerful NAPALM feature called compliance report or deployment validation. To get a compliance report you need to write a YAML file describing the desired state of the device and tell NAPALM to use it against the device with a `compliance_report` method.

Summary

All of the described tools have their advantages and use cases. I would recommend starting with more high-level tools such as NAPALM, Ansible, or Nornir.

Text editors

A text editor is a piece of software you will spend most of your time with while automating networks. Here I would like to make an overview of the most popular modern text editors.

VS Code

Good or bad, but today's text editor market is clearly dominated by Visual Studio Code. It has a great and ever-expanding collection of plugins, nice UI, built-in Git support, intelligent code completion, you name it.

VS Code is a free and open-source text editor built on Electron and owned by Microsoft. It was initially released in 2015.

Atom

Atom is another highly customizable open-source text editor created by GitHub. Since GitHub was acquired by Microsoft, Atom now is also a Microsoft product.

Atom also is free, open-source, and built on Electron. It was initially released in 2014.

PyCharm

PyCharm is a full-blown Python IDE by JetBrains. I've heard a lot of praise towards it in the context of Python development, but never tried it myself. PyCharm is shipped in two versions: full-featured Professional ($89/year subscription license) and less functional but free Community Edition.

PyCharm was initially released in 2010.

Sublime Text

Sublime is the oldest text editor on the list. It has some great features to itself like multiline editing and "Go To Anything" command which allows to quickly jump to the specific part of the text in any open tab. It also can be extended with plugins, but the package manager is not included by default and you'll have to install it manually first.

Sublime Text is a proprietary paid software written in C++ and initially released in 2008. It has a 30 day trial period. After that, you will be kindly reminded from time to time to buy an $80 license.

Section to Track my Finance related Work

Python Script to Backtest DCA

```
import os
import pandas as pd
import yfinance as yf
from datetime import datetime, timedelta

def get_trading_dates(start_date, end_date):
    trading_dates = pd.bdate_range(start_date, end_date, freq='W')
    return trading_dates

def record_data(date, ETF_price, shares_purchased, total_investment,
total_shares, weekly_investment, ROI=None, final_results=False):
    return {
        'date': date,
        'ETF_price': ETF_price,
        'shares_purchased': shares_purchased,
        'total_investment': total_investment,
        'total_shares': total_shares,
        'weekly_investment': weekly_investment,
        'ROI': ROI,
        'final_results': final_results
    }

def calculate_final_results(data_records, weekly_investment):
    last_record = data_records[-1]
    total_investment = last_record['total_investment']
    total_shares = last_record['total_shares']
    ETF_price = last_record['ETF_price']
    final_portfolio_value = total_shares * ETF_price

    ROI = (final_portfolio_value - total_investment) / total_investment

    max_portfolio_value = 0
    max_drawdown = 0
    for record in data_records:
        portfolio_value = record['total_shares'] * record['ETF_price']
        max_portfolio_value = max(max_portfolio_value, portfolio_value)
        drawdown = (max_portfolio_value - portfolio_value) /
max_portfolio_value
        max_drawdown = max(max_drawdown, drawdown)

    return {
        'total_investment': total_investment,
        'total_shares': total_shares,
        'final_portfolio_value': final_portfolio_value,
        'ROI': ROI,
        'max_drawdown': max_drawdown,
        'weekly_investment': weekly_investment
    }

def DCA(ETF_ticker, start_date, end_date, weekly_investment):
    trading_dates = get_trading_dates(start_date, end_date)
```