gluware®
Intelligent Network Automation

Progressing Network Automation

Moving from Script and Playbook to Model-Driven, Intent-based Network Automation, and Orchestration

e-Book

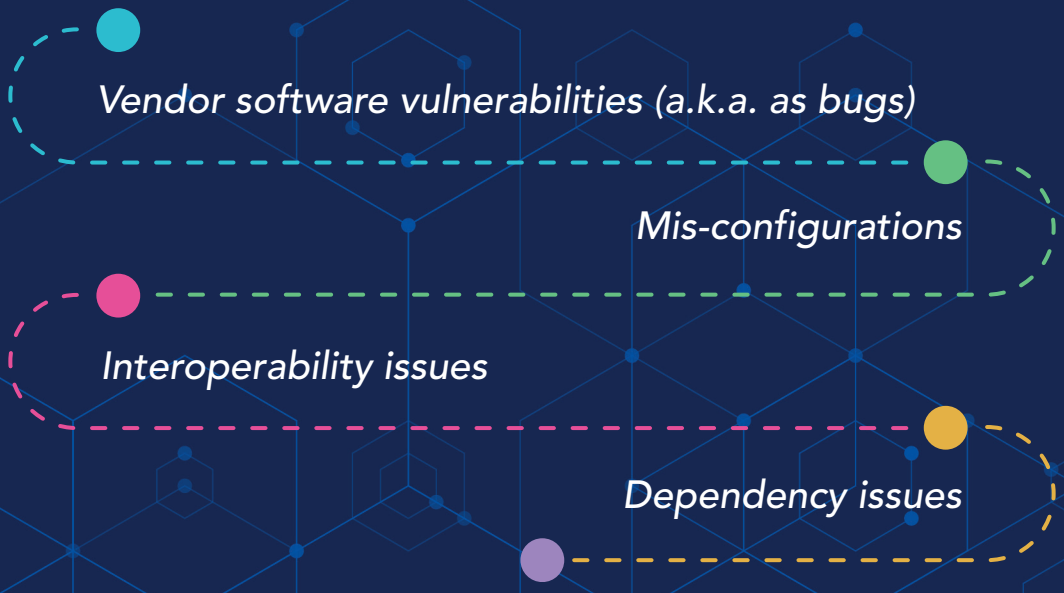
The Current State of NetOps

Internet Protocol (IP) networks were designed to be distributed and highly fault tolerant to disruptions by being able to dynamically "route" network traffic around outages or connectivity problems.

The network layer equipment, mostly made up of routers and switches, were designed to be configured as stand-alone devices, one-by-one, using a low-level human readable format called Command-Line-Interface (CLI), which at the time was much simpler than other options that had come before it. Over the past 20+ years, nearly all computer networks are built based on IP technologies. While protocol features and hardware had evolved, most network configuration changes are still being performed in the same way – manual, one-by-one. As function specific devices were added like firewalls, load balancers, WAN optimizers, and wireless LAN controllers the "defacto" configuration method via CLI propagated to these devices as well. Network devices are a significant capital expense (CAPEX) and are typically deployed for a minimum of five years, or much longer in most cases. This creates a constant legacy of existing, older devices known as a "brownfield" network. Brand new deployments, of new campus switches or a new data center, are known as "greenfield" networks. As new network functions are deployed and new vendors are introduced the unique CLI/semantic required for each vendor causes significant operational expense (OPEX) increase for the Network Operations Center (NOC) to develop and maintain the skill set required.

As protocols and features have been added over the years, the configuration has become significantly more complex and a highly specialized skill set is required for initial and on-going configuration management. Due to the complication and cost of an outage, network operators are very change averse and typically space changes out and only perform them during a scheduled maintenance window where a minor outage could be tolerated.

Changes To The Network Can Introduce Risk From



Source: Cisco

While the "Software Defined Networking" (SDN) movement intended to change the way networks are configured and enable a programmable network layer, controlled by adding a software layer it has had limited success and is slow to become mainstream due to the investment in the legacy network and the significant changes required to adopt SDN. SDN also is early in its maturity with limited standardization and many different definitions of what it actually is. SDN in the data center is primarily overlay tunneling that still relies on the existing underlay network. SDN in the WAN, known as SD-WAN is very vendor specific and generally requires a refresh of the branch and headend devices, which is a significant investment for a new technology that is still developing rapidly. Most SDN technologies do introduce a software-based network controller which provides centralized control of configuration and policy management along with various monitoring capabilities. Organizations adopting SDN in its early state often have to build separate teams within their organizations to support operations along with the traditional legacy management teams.

Team

The Network Operations (NetOps) team is typically made up of several roles including Network Architects, Network Engineers, and Network Operations Engineers/Technicians. Each has a unique skill set contributing to the overall operations and management of the network.

Network Architect

- Network design and standardization
- Protocol expertise
- Cross-platform/vendor expertise
- Solution lifecycle management

Network Engineer

- Protocol expertise
- Vendor/Platform specialization
- Testing and validation
- Operating/managing a network

Network Operations Engineer/Technician

- Device/Service configuration
- Device/Service troubleshooting
- Management and ticketing systems
- Vendor/Platform specialization

NetOps teams are known to have a long list of tools to manage the network including monitoring, ticketing, asset management, software and configuration management and more. While some tools are based on well-known and adopted technologies, like Simple Network Management Protocol (SNMP) used primarily for monitoring, others are very fragmented and vendor specific. Lack of end-to-end network visibility, shortage of skill set and fragmented management tools are always among the top three complaints of NetOps teams.

Tools

One of the areas to point out is configuration management. The current generation of configuration management tools are single vendor (or very single vendor focused) or are "bulk update" tools which can push configuration changes with very little intelligence or verification, which can introduce more problems than they solve. This results in a significant challenge to implement and enforce policy across the network when many changes are being performed manually and ad-hoc in an attempt to quickly resolve issues for a quick fix.

Enterprise IT is changing dramatically, largely driven by compute virtualization and cloud-based services. The changes in application development are driving the need for more agility and support for more frequent changes to the required networking services performed by NetOps.

Implementing change in NetOps is difficult for most IT organizations since they are operating a business-critical network infrastructure and prefer not to take risks that could cause expensive and potentially embarrassing outages.

For change to be successful, it has to be driven across

3 Main Categories
People · Products · Process

1

People

Most IT organizations have network engineers or specialists who have spent years becoming network experts, which includes knowledge of the network protocols and various vendor CLI and semantics to configure the devices. There is generally a very specific path network engineers follow to become network experts, moving from operations to engineering to design and architecture. Engineers have become very comfortable developing the low-level, vendor-specific syntax to configure the network layer. Until recently, a programming background was not a requirement for a network engineer. With the increase in scale 1 and complexity manually configuring all network nodes is no longer an option to keep up with the business needs. Now, management is deciding how much time and money to invest in training their network team to be programmers. The engineers are also faced with a career decision – whether to remain a network expert, potentially becoming obsolete, or expanding their skill set to learn programming and application programming interfaces (APIs). In addition, the path to becoming a programmer for networks is not as clear as engineering since there are a plethora of tools and languages each requiring hundreds if not thousands of hours to learn.

2 Products

As organizations dive into the world of network programming, they may start with investigating SDN technology since it has promised to make the network programmable. The real driver is often not necessary to move to SDN but to have a mechanism to automate manual tasks and keep up with the business demands since the days of static network configurations are becoming a thing of the past. The gap here is that most SDN solutions require new networking hardware or are an over-the-top software solution still leaving the underlying network to manage. Enterprise IT wants the benefit of SDN without needing to deploy and manage a new network be it physical or virtual.

Looking into the landscape of network automation tools available, there currently seem to be two ends of the spectrum and not much in between. At one end, you have programmer/developer-oriented platforms, which require knowledge of a programming language and the environment involving dedicated servers, databases, code repositories and more. On top of that, each area can have significant specialization for "front-end" user interface design, middleware and back-end (database) systems. At the other end of the spectrum, you have network automation tools, which are point solutions providing a very specific function like automating a QoS policy. These "tools" are purpose-built and have very rigid functionality – not likely to meet the ongoing and changing needs of most IT organizations. Some point solutions also can even build full configuration files for specific deployments like a spine-leaf data center, but again they have very specific design limitations and must be a perfect fit.

The middle of the spectrum goes largely unserved. This would be a platform that offers programmability to be customized to a unique network solution but also provides some level of turn-key solutions to address common pain points and tasks that are done manually today. Current SDN solutions are primarily providing an API on top of a network controller, which requires programming to interface with the API since there is a lack of true network applications, especially when it comes to multi-vendor networks.

3

Process

Finally, it comes down to the process and the overall motivation for change to embrace network automation. If the organization has a lot of resources and is able to hire a development team, then the developer-oriented solutions could be a good fit. If the organization has some specific pain and no development resources and the point solutions available will address it, then that could be a good fit however short lived once the network changes. This is often the case and IT teams have many tools that largely go unused. The key is to take a long-term, strategic approach to the problem and decide the direction so that the cost involved to hire, train and purchase software is returning sustained value to the organization. What are best practices that companies with lean IT can follow when looking to solve business pain points through network automation? Why automate the network?

Why Automate the Network?

Efficiency Gains	Productivity Improvements	Cost Reduction
Reduce human touch	Increased speed/agility (of configuration/deployment)	Eliminate or reduce current spend
Reduce errors (misconfigurations)	Improved quality	Eliminate or reduce future spend
Automate redundant tasks	Deliver on more change requests (volume of change)	Reduce/re-purpose headcount
Enable shifting of resources to strategic projects	Increased engineer productivity (ability to manage large network)	

Automation provides standard benefits such as increased efficiencies and performing tasks faster, cheaper and more reliably versus manually. The issue with network automation is that most current network engineers do not want to become programmers and think that is the only path. Looking at the automation landscape, there are three main approaches to investigate: script based, script/playbook plus an automation framework, and a new approach which is data-model driven and enables automation without programming.

Starting the Network Automation Journey

The automation journey begins with an assessment to determine the current state, desired state and how to acheive it. This includes asking:

What do I need to automate?

- Do I have a current network inventory including vendors, platforms, OS versions...etc.?
- For each platform what is the current state of the configuration (compared to any existing policy/ standard)?

What teams/network domains do I need to consider?

- Campus LAN, WAN, Data Center…etc.

What should I automate first?

- Is there a pressing business demand/strategic project to align with?
- Merger/acquisitions
- Migration or enabling a Software-as-a-Service (SaaS)
- Security vulnerability
- What are the top issues raised to NetOps? Is there a ticket system to gather this data?
- What type of changes has resulted in the inconsistent or error-prone configuration?
- What are the most time-consuming work tasks NetOps is currently performing manually?
- Are there compliance (internal or external) requirements to align with?
- Are there any current (known) security vulnerabilities that need to be addressed, possibly via an OS upgrade?

What tools, platforms, products should I use?

- Have any tools, solutions been used and what was the result?
- What is the best solution available for the current issue and top priorities going forward?
- What have the shortcomings been of the legacy/existing toolset?
- What is the time to learn, onboard and maintain the products being considered?

For each initiative, what is the return-on-investment (ROI)?

This must be considered if dedicating significant man-hours and/or purchasing a product to implement the automation.

Does automation in your organization means a low-value utility for simple changes, or does it consider the lifecycle automation of devices/services?

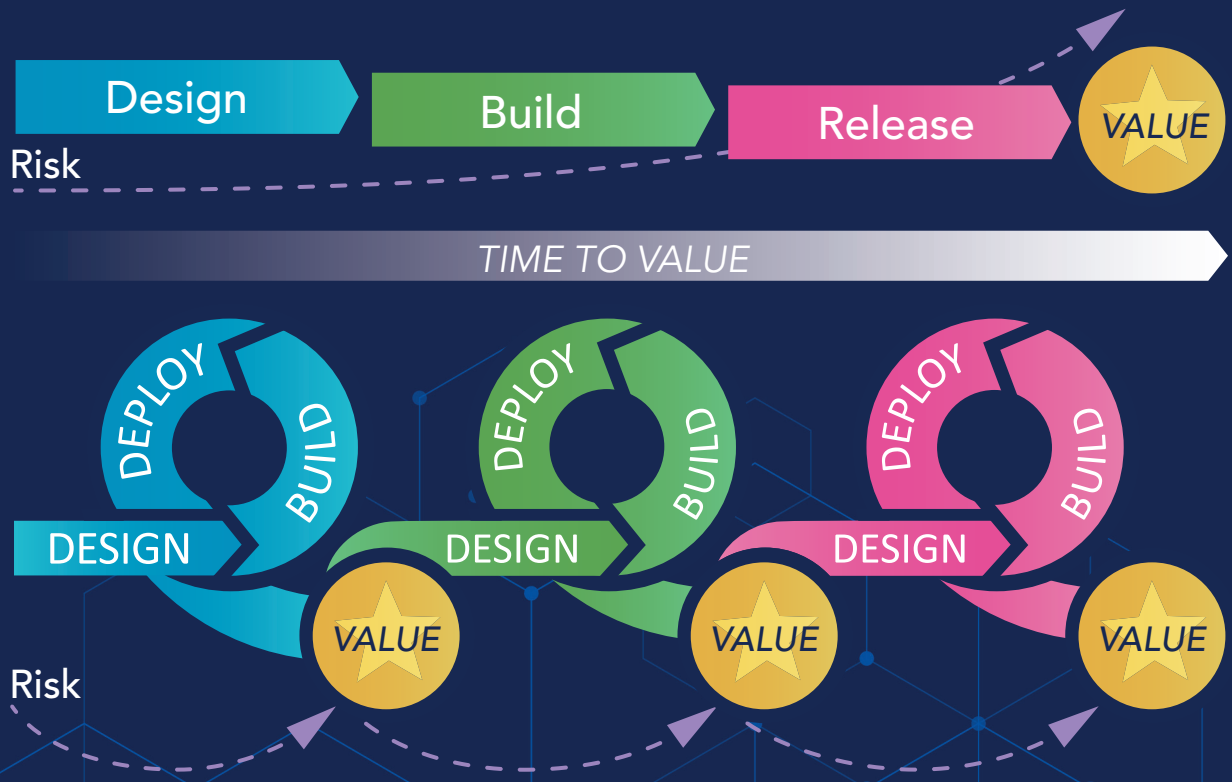
Is The Need For Automation

Strategic solutions are more likely to get sponsorship from upper management and get funding for products and training to accomplish the task within a well-defined time frame. A strategic approach should always consider the lifecycle of the product/service. When a device is initially deployed in the network it is referred to as "Day Zero". That initial configuration is often straightforward to automate since it is new. Day 1, day 2 …. day N are the ongoing changes that occur. Some of these changes are well designed and tested, other changes are based on operations decision to address a problem in the network – which can be a quick fix, but potentially over time takes that device way out of compliance with the standard/approved configuration. Proper automation and change management control will eliminate one-off changes which create problems further in the lifecycle of managing those devices. When implementing automation, it is critical to think about the lifecycle management of the network layer devices and the services being provided to the business. As the application teams move to a DevOps model for a more rapid, agile deployment model the NetOps team will also have to implement a dramatic change in their process to keep up with the business needs.

Moving NetOps towards the DevOps Model

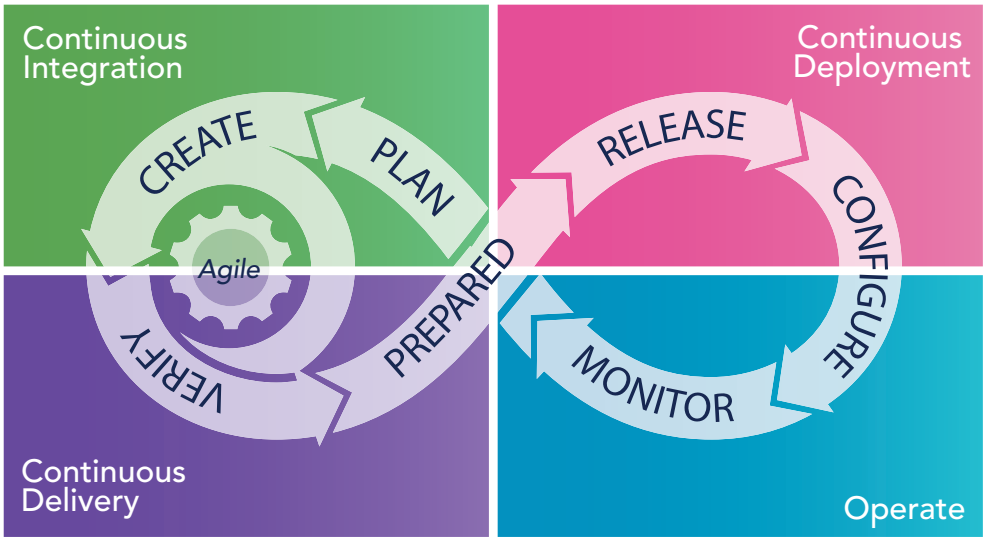
DevOps evolved, out of necessity, from the software world. With the Waterfall development model, there was a lengthy period of product definition, and developers usually worked over a several-month period of testing and essentially developed a solution that was already out of date. Agile was the next evolution in software development, in which there is an understanding that requirements are going to frequently change. The idea is to get something out the door that meets the requirements and then keep iterating and adding to it. This started the trend of what's now known as DevOps, which is a streamlining of the process from development to operations. With DevOps, you have to be continually developing, testing, integrating and deploying changes. An abbreviation sometimes used for DevOps is CI/CD, continuous integration/continuous deployment. It's a constant cycle of innovation rather than a monolithic process. This enables agility and responsiveness to the changing business needs.

A similar phenomenon is happening in networking. You can't take six months to get a feature out the door anymore. If your terms of service or a business line need something in the network, you need to get it to the network as soon as possible. That's where networking engineers are at right now. This applies to networking from the deployment and provisioning of new services and features as well as the ongoing changes needed to routing, security, QoS and other policies on the network.

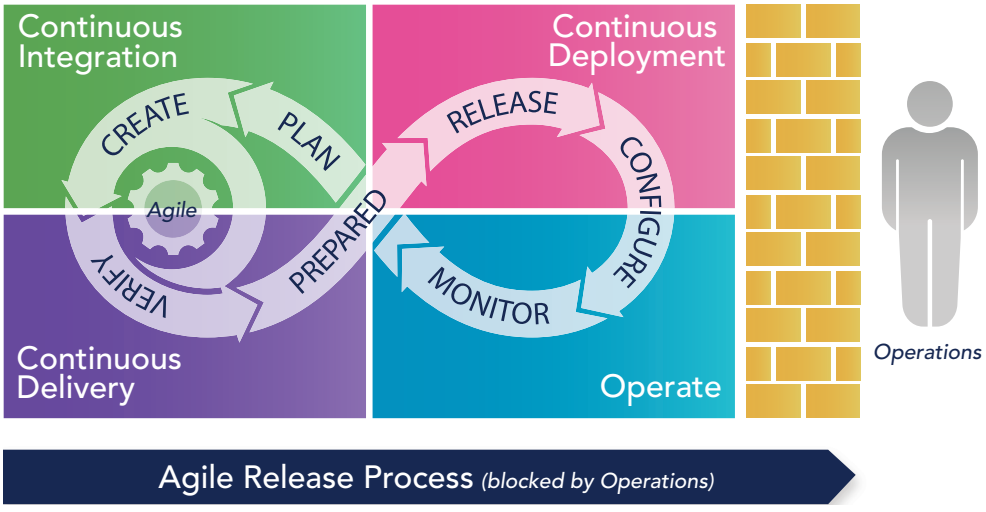


Moving to an Agile Development Enables Faster Time to Value

The CI/CD Model is in Constant Motion to Generate Value to the Organization



Operations Cannot be the Barrier for the Agile Development to be Successful



Network Automation 1.0: Scripting (D.I.Y.)

Scripting is a general-purpose programming language that can be used for finance tracking, sports teams management, automating a home, automating different tasks on a computer, or creating a server configuration management tool (more on this later). The flexibility of scripting makes it a viable choice for automating a network.

The first step in choosing scripting is selecting a language. Some of the mainstream scripting languages to choose from (in no particular order) are Perl, Python, PHP, Ruby, and JavaScript. When making a language choice, understanding the ecosystem around that language is important. Does the language have libraries, frameworks, and a community to support the kind of application being created?

Python has emerged, in recent years, as a dominant choice for automating networks. Python has libraries like Paramiko (to communicate via SSH to devices), Netmiko (built on Paramiko that has specific device support for show commands, configuration command, etc.). It has frameworks for automated testing of scripts. There are community resources (blog posts, videos, and podcasts) available around network automation with Python.

Writing a script is not that hard. A motivated technical person could do it within a week or so for a "simple" network configuration task. This depends on a good understanding of the task and the knowledge of the script writer of the scripting language, used libraries, and used frameworks.

- If the goal is to have scripts usable beyond just a person writing it then things get more complicated. The script writer would have to code the script with more logic to allow other people to use it.
- If the script has parameters that control the actions of the script, that the script user enters, then logic has to be written to validate that input.
- If the script is talking to multiple networking devices then it would be desirable to have a external data source for the addresses and connection information for the devices. That way the devices the script acts on could change without the script logic being changed.
- The script would need some output to indicate its progress, if it errored where the error was, and if it successfully completed. A standard way should be created so each script doesn't have its own format.
- The script would need to be documented in some way (at the very least that it exists and what functionality it provides) so that other people could use it. The documentation should include what parameters the script takes.
- If the script was complicated enough then the script writer might have to make a "preview" mode to show the script user what would happen for the parameters they entered without it actually happening.

Once this reusable script is created then general software development questions have to be answered for managing the scripts themselves:

Source Control

- Where is the script stored?
- How do other authorized people modify the script for changes and bugs with the script?
- How do you track changes to the script?

Issue/Change Tracking

- How do you collect and document bugs and future changes in the scripts so they are managed and you know the status when they get fixed?

Release Management

- Scripts should be versioned when made available so it is known what fixes/changes went into the script.
- Should scripts be tested and verified in some way once they are released?

Using the Scripts

- Is there a Graphical User Interface created for running scripts? Is it all command line driven?
- Should there be some kind of authorization for running the script?
- Where should the scripts run from? The user's PC or a common server?
- Is there a way to share device address and connection information between scripts?
    - How does this device information get updated?
    - Is there security for this data?

Python

While it can take months to investigate and begin to build the skill set in the items listed above, most engineers will look to leverage what is most used in the industry and for network automation, the language of choice is currently Python.

Over the last ten or so years there has been a lot of vendors and community development around using the Python language for network automation. As a result, there are vendor and community-supported Python libraries to leverage and accelerate the creation of a script. The Python approach will involve a significant amount of coding, but far less than it ever used to be for network automation. Beyond the jump start you get with the available libraries, Python is also known to be a good beginner language since it is human readable and does not require the pre-definition of variable types and functions, which reduces the coding effort.

In short, what is Python and why is it the language of choice for network automation?

The Python Foundation provided this Executive Summary:

> *"Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level, built-in data structure, combined with dynamic typing and dynamic binding, makes it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy-to-learn syntax emphasizes readability and therefore reduces the cost of program maintenance."*

Python is an interpreted language (it does not need to be compiled), so the speed of write, test, and edit of the program process is greatly increased. For simple scripts, if your script fails, it can easily be debugged and edited to resolve the issue and move forward. Using the interpreter also means that Python is easily ported to a different type of operating system. For example, Python can be developed on a Windows machine and it can be easily ported to run on Linux or MAC OSes. Python is also object-oriented so there is a high degree of re-use that can be achieved. Python installs with a standard set of libraries and can easily import functions and modules to meet the use case desired.

Python Skill Sets Required

- Python Interpreter (shell)
- Data types
- Conditional logic
- Containment
- Functions and methods
- File management
- Creating programs
- Modules and packages

Python offers an option for a NetOps person to become a scripter and achieve some level of automation without becoming a full-blown developer since it is easier to use than most other languages.

A challenge with D.I.Y. scripting is that often a small task is automated and then that successful script gets added to and extended until the simple script is now quite complex. To be successful with scripting requires good code hygiene and developing a master plan to add functionality while keeping specific functions working.

An important note about Python versions is that many users are in a transition from Python v2.x to Python v3.x and Python 3 is not backward compatible; however, the versions can coexist. Python v2.7 release has gone end-of-life and is only getting security updates.

Using Python to Interact with Network Devices

As already covered, most (legacy) network devices are configured via CLI which requires a terminal session (connection) with each device. While it is possible to use the Telnet protocol, pretty much all devices in production will use SSH, a secure terminal protocol. Using SSH, commands are passed to the device over a persistent connection and the device interprets them and responds with human readable text output. SSH does not structure or encode the data, which is not ideal when programming, but most legacy devices do not provide a programmatic interface like an API. Python offers third-party libraries to help add this functionality. A few popular libraries to add are Pexpect, Paramiko, and Netmiko.

Python Pexpect Library

Pexpect is a pure Python module for spawning child applications, controlling them, and responding to expected patterns in their output. Pexpect allows your script to spawn a child application and control it as if a human were typing commands.

Pexpect launches or spawns another process and watches over it in order to control the interaction. It can be used to spawn a telnet (or SSH) session to a network device, send commands, wait for an expected output and send additional commands.

Python Paramiko Library

Paramiko is a Python implementation of the SSHv2 protocol. It simplifies the SSHv2 interaction with the remote device. Paramiko only supports SSHv2 and provides both client and server operations. It has a hard dependency on the Cryptography library which also must be loaded to use it.

Python Netmiko Library

The Netmiko library is a popular open source Python library that simplifies SSH management to network devices. It is built on top of the Paramiko library. With support for over two dozen device types, it is one of the most widely used libraries for SSH to simplify the device connectivity and commands used.

These libraries among many others can be found at
https://pypi.org

They can be dynamically loaded into Python via an integrated software distribution called Python Package Index (PyPI) and are loaded via the pip command. In addition, it is common to load Python packages from source (GitHub, a public code repository and version control platform).

In the context of scripts that interact with network devices, this essentially means repeatable, reliable, predictable interaction with the network device is a requirement for a successful script.

Using the Python plus Pexpect/Paramiko approach emulates a user on a terminal connection and will enable a script to interact with a device, but since it is working with non-structured data and performing a "screen scrape" to capture the output of the CLI, there is a chance you could have inconsistent results if there is any variation on the output. For example, if the output is many lines of characters and there are extra spaces or line breaks this can cause issues with the desired result and create inconsistencies.

If the goal is to make reliable changes on many devices, say hundreds or thousands, another approach beyond scripting will be required to ensure consistency in the results.

Data Models and Templates

Python is an extremely flexible language that can work with many data model formats and templating languages. For data modeling, the popular choices are XML, JSON, YAML, and YANG. In general, all have pros and cons but certainly can be helpful to leverage when developing automation for network devices.

Templating is a common and well-known practice in network engineering, although most templates are built with native CLI or even using Excel. When working with Python the most popular templating language is Jinja since it is built for Python. Working with Jinja can enable the insertion of device-specific variables into a configuration file when generating a unique configuration for many devices. Jinja is another 3rd party package which can be installed using the pip command from PyPI.

An example leveraging Jinja looks like this to introduce variables into a configuration template

```
interface {{ interface.name }}
description {{ interface.description }}
switchport access vlan {{ interface.vlan }}
switchport mode access
```

Often, when using Python for a network automation script, the best practice will be to import data from another source (like a data model or template) and use coding concepts, like loops and conditionals to generate the desired native CLI configuration which is "rendered" upon script execution. It is a common practice to keep Python syntax separate from the actual data files (not embed it in the script), instead having it defined in a separate file, most likely using a data modeling language of choice. Many network vendors are already beginning to support data model formats like JSON, XML, and YANG.

Scripting with Python Pros & Cons Summary

Pros	Cons
Free software download	Significant skill development required (time and training required)
Open source and strong community	Scripts will vary significantly based on author, design choices and skill set
Flexible language to automate networking (multi-vendor) and full stack (compute, storage)	Scripts are often single purpose and single-use, requiring edits to the script the next time they are needed to make a change
Simple tasks can be automated fairly quickly with basic scripts with the required skill set	Gets complex quickly to automate complex networking feature (like CERTs, Tunnels, Routing, QoS…etc.)
Python is easier than most other languages to get started with	Difficult to maintain (dependencies, software versions, script edit revisions)
Many network vendors have example libraries to get started	Difficult to transition to other users/operators
	Lack of a UI and other integrated functions NetOps has gotten used to with other management tools

Network Automation 2.0: Server Configuration Management Tools (Scripts+Playbooks)

Since the move for DevOps to CI/CD has had some success in most Enterprise IT organizations, often when looking for a NetOps solution one of these server configuration management tools are adapted to meet their requirement. Some of the more well-known server configuration management tools (in no particular order) are Puppet, Chef, Ansible, and Salt. Some now have support for some network configuration. These tools are purpose built to automate infrastructure changes. They each have their own architecture and terminology.

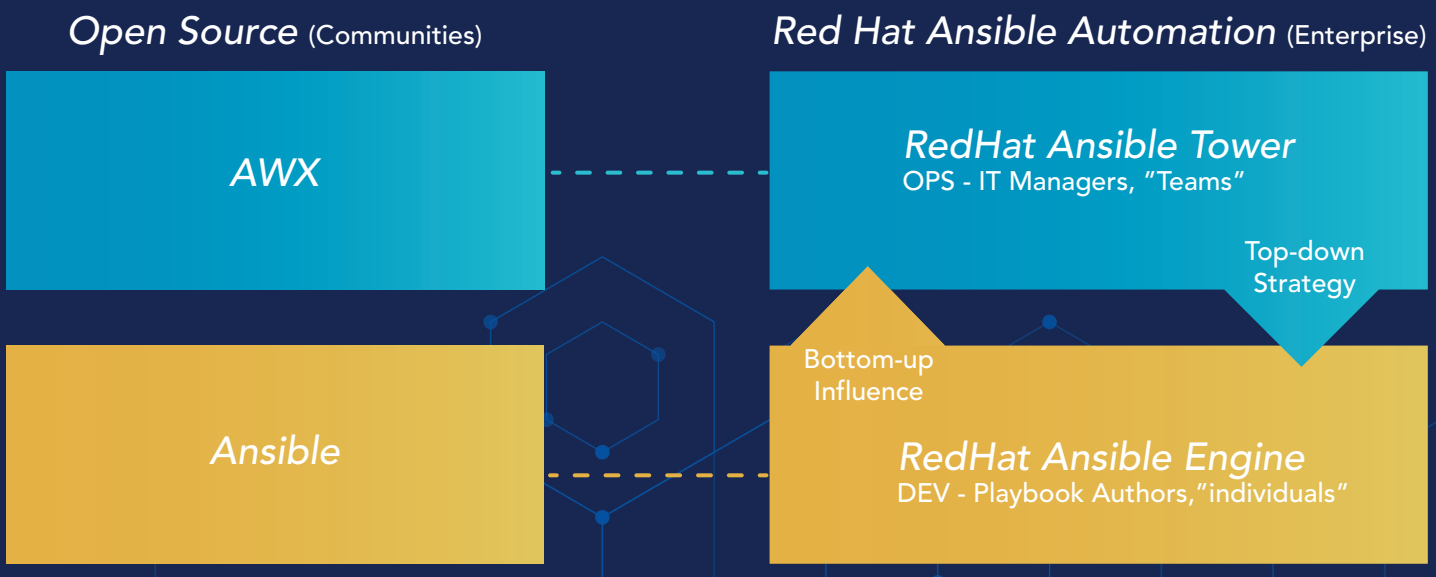
A server is a machine with an Operating System (OS) that runs third party application software. The application software is the value of that server. A server with just an OS doesn't do anything but warms a room. All application software has their own configuration files and language to control how it runs. There can be many servers with many running application software instances. This is the problem server configuration management tools solve. The server configuration management tools define their own language to install/configure/upgrade applications running on the servers (Cookbooks in Puppet, Recipes in Chef, Playbooks in Ansible, and Formulas in Salt). This gives operations a tool to manage their infrastructure with a uniform language that simplifies operations.

ANSIBLE

Ansible Skill Sets Required

- Basic Linux/Unix
- Ansible packaging/options
- YAML
  (language for Ansible Playbooks)
- Jinja2
  (language for templating)
- Python
  (to extend Ansible)
- Data types
- Variables
- Conditionals
- Loops
- Blocks
- Handlers
- Playbook roles
- Plays, tasks, modules, parameters
- Repositories (GitHub)

Ansible Components



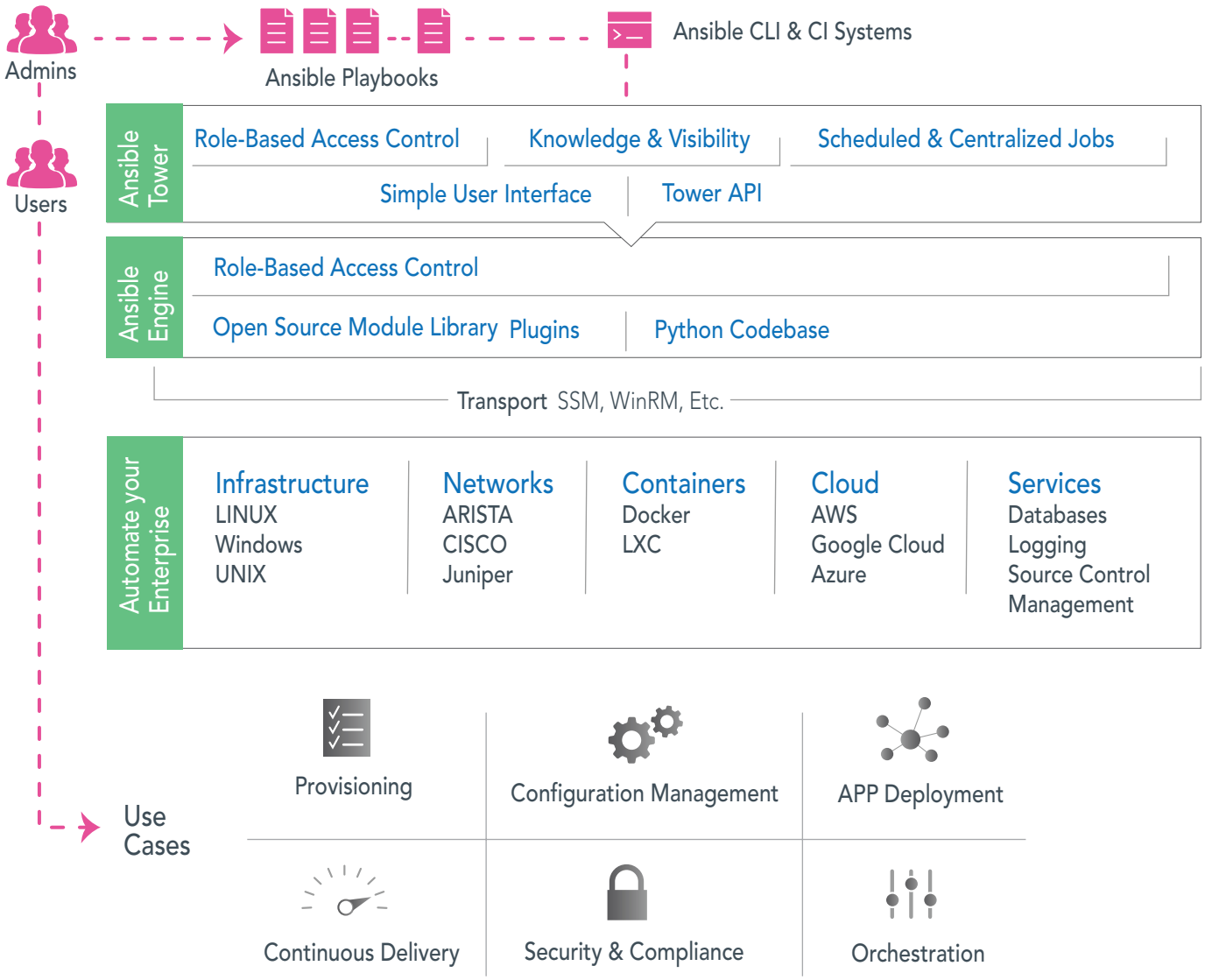
Ansible Open Source and Commercial Offerings

Open Source

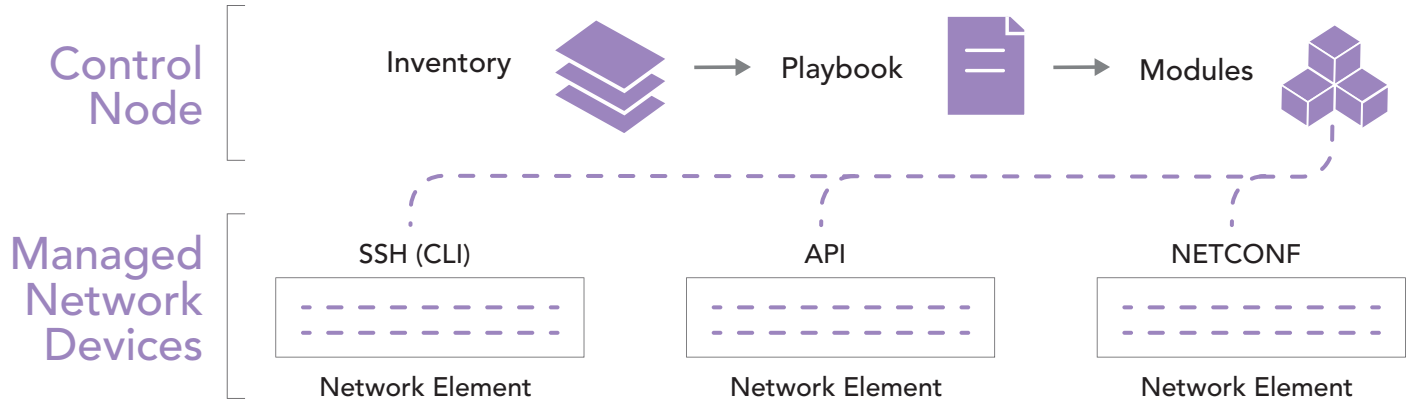
- Ansible (core) is the open source distribution of the automation software platform.
- Ansible AWX is the open source version for operational environments.

Commercial

- Red Hat Ansible Engine is the Enterprise version of Ansible (core) which is the same code base and adds support instead of just user communities.
- Red Hat Ansible Tower packages feature for Enterprise IT Operations teams which include adding Role Based Access Control (RBAC), secure storage for network credentials and other features.



Ansible Architecture



Ansible Elements to Manage Network Devices

To get started with Ansible the first step is installing Control Node, which must be a Linux or Mac OSX system and has a dependency of a Python installation. Ansible then requires three main components to begin automating network devices:

Inventory This is a plain text file that lists the target nodes which can include specific devices (hosts) and groups.

Playbook(s) are Ansible's configuration, deployment, and orchestration language. These define the instructions to describe what actions/automation is to be done to the network elements using the Modules. These are written using the YAML markup language format and are designed to be human readable. A playbook consists of one or more plays. Each Play contains one or more tasks. Playbooks can be written in standard YAML or an Ansible derivation of YAML.

Modules are device-specific plug-ins that handle the execution of the playbook on the device.

As depicted in the image, Ansible has integrated support for transport protocols to connect to network devices including SSH/CLI, API, and NETCONF.

Unlike using Ansible to automate servers, when using it for network automation it is configured to run in local mode where it connects to itself and runs Python code locally then connects via SSH to network devices to send CLI commands.

Idempotent

The property of certain operations in mathematics and computer science, that can be applied multiple times without changing the result beyond the initial application.

Source: Wikipedia

Ansible aims to be idempotent. When executing a playbook, the existing device configuration will be checked first, and the playbook will only execute if the changes do not exist. The reliability largely depends on the vendor module and the quality and content of the playbook. Significant testing is required to ensure consistency and reliability at scale.

Variable and Templates

Variables can be defined in the inventory file as well as the playbook. Ansible uses the Jinja2 templating language. For network device automation a good example of the variables in a playbook are for the transport protocol:

```
vars:
    cli:
    host: "{{ cisco_router }}"
    username: "{{ username }}"
    password: "{{ password }}"
    transport: cli
```

Network Modules

The majority, if not all, of the Network Modules, have been contributed by the vendors. While they are a good place to get started, many will involve the learning curve to develop, test and eventually deploy successful automation using playbooks.

For example, Cisco provides modules for IOS, IOS-XR, and NXOS. The "ios_config" module is used to send configuration-level commands to Cisco IOS devices. See the example playbook snapshot below:

```
- name: configure top-level configuration
ios_config:
    lines: hostname {{ inventory_hostname }}

- name: configure interface settings
ios_config:
  lines:
    - description test interface
    - ip address 172.31.1.1 255.255.255.0
    parents: interface Ethernet1
```

```
- name: configure ip helpers on multiple
interfaces
ios_config:
    lines:
    - ip helper-address 172.26.1.10
    - ip helper-address 172.26.3.8
    parents: "{{ item }}"

with_items:
    - interface Ethernet1
    - interface Ethernet2
    - interface GigabitEthernet1

- name: load new acl into device
ios_config:
    lines:
    - 10 permit ip host 1.1.1.1 any log
    - 20 permit ip host 2.2.2.2 any log
    - 30 permit ip host 3.3.3.3 any log
    - 40 permit ip host 4.4.4.4 any log
    - 50 permit ip host 5.5.5.5 any log
    parents: ip access-list extended test
    before: no ip access-list extended test
    match: exact
```

As you can see from the above example, the configuration is similar to native IOS, but there are specific structures and syntax that must be added to achieve the desired result. A network engineer who has CLI/syntax skill set must also learn the module features and semantic to adapt the native CLI into the playbook format.

Ansible Usage for Network Automation

Ansible is a common choice for automating network infrastructure. This is mainly because of its agentless design. An agentless design means Ansible doesn't require a piece of software to be installed in the device being managed. Many network devices have no capability to install or run third party software. So, this makes Ansible a good fit.

- Create playbooks to auto-generate multi-vendor configurations
- Create playbooks with conditional elements to deploy configuration based on existing state
- Create playbooks to read and gather data (configuration and operational) from network devices

The challenges around creating reusable scripts goes away with a server configuration management tool

- The tool takes care of parameter input validation.
- Ansible defines a separate inventory file independent of its Playbook. So, this is how device address and connection information is handled.
- The tool has a standard way to show progress, error information and if it succeeded.
- There is lots of documentation for using these tools.

Things that still have to be considered with Ansible

- It would still be desirable to put Ansible Playbooks into source control to track its changes and be able to rollback.

Using the playbooks

- Red Hat's Ansible Tower is an enterprise product that users can purchase to manage the running of Playbooks.
- If Ansible Tower is not used then the same issues with using scripts applies to Ansible.

Pros And Cons

Ansible Pros

- Free software downloads to start
- Open source option, strong community, and vendor contributed modules
- Leverages Python and YAML, well-established languages to define and extend the functionality
- The breadth of modules to support compute, network, containers, and cloud
- Simple tasks can be automated fairly quickly with basic playbooks with the required skill set

Ansible Cons

- Significant skill development required (time and training required)
- Playbooks will vary significantly based on author, design choices and skill set
- Gets complex quickly to automate complex networking feature (like CERTs, Tunnels, Routing, QoS…etc)
- Difficult to maintain
- Significant cost added to use Enterprise Ansible Engine and Ansible Tower
- Difficult to transition to other users/ operators
- Must add Tower to provide the user interface and integrated functions NetOps has gotten used to with other management tools (with support)

Enterprise-Grade Capabilities

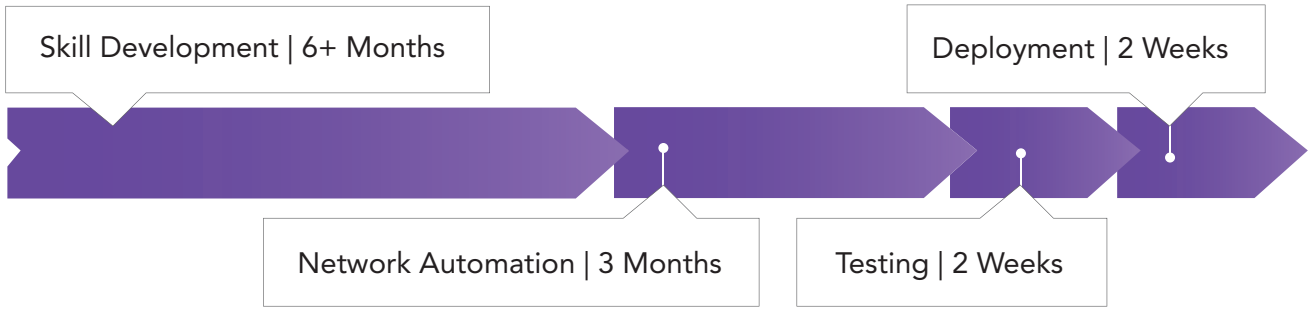
Gluware is built to simplify network automation and minimize the time to value. Now on its third generation and built over many years, it provides a robust feature set to meet the needs of very large enterprise customers managing complex networks scaling into the tens of thousands.

Automation Platform Checklist

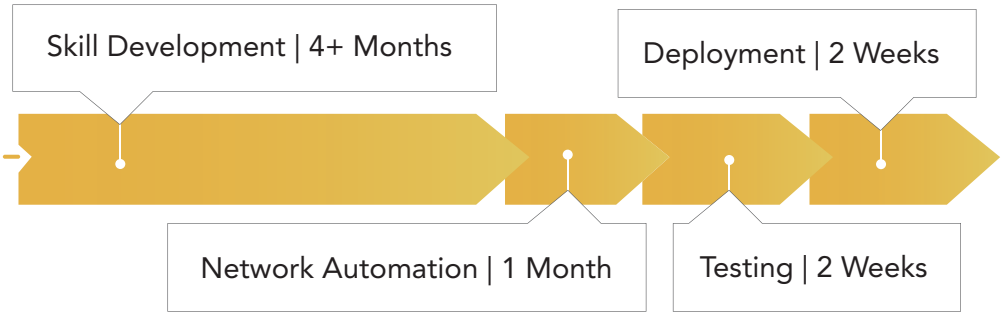
- ✓ Onboard network feature and define policy in hours with no skill set gap
- ✓ Vendor agnostic feature abstraction or leverage native CLI
- ✓ Automate the brownfield existing network automating as many or as few network features
- ✓ Data-model driven and declarative orchestration engine
- ✓ Support for legacy network configuration and change management (NCCM)
- ✓ Multi-domain – supporting automating the LAN, WAN, data center and more
- ✓ Multi-user with LDAP integration and rights/roles management
- ✓ Multi-tenant enabling management of many administratively separate networks
- ✓ Ability to extend support to new, emerging virtual infrastructures leveraging SDN and Network Functions
- ✓ Virtualization (NFV)
- ✓ A platform that can help the IT organization adhere to corporate and government compliance policies, auditing standards and regulations
- ✓ Ability to perform secure and reliable changes to the network, across multi-vendor network devices
- ✓ Identifiable return on investment (ROI) to ensure alignment with the business needs
- ✓ Ability to implement automation quickly, with limited training and onboarding time
- ✓ An extensible platform which can grow and change with evolving business needs

Time to Value Comparison

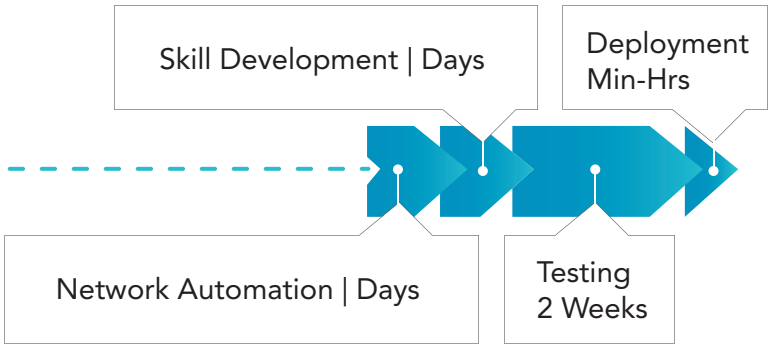
Python Time to Value per Project (est. 10+ months)



Ansible Time to Value per Project (est. 6+ months)



Gluware Time to Value per Project (est. >1 month)



Conclusion

The only thing that's consistent in the network is change. Business needs are changing and the way you're implementing features on the network is changing, so the infrastructure you need at the network layer to move from Development to Operations needs to resemble the software world's movement to DevOps and its continuous integration cycle.

Network engineers are not software developers, nor should they have to pretend to be. Engineers today need an orchestration platform that enables the full integration of modeling, building, testing, deploying to production, validating and enabling the continuous integration cycle. This model reduces friction between Development and Operations. Whereas formerly these teams would be at odds, development and operations work closely together in a DevOps model. Operations provide feedback to Development, and they have a relationship that minimizes overhead, inefficiencies and the process of getting changes out to production.

As organizations look to increase automation the decision will largely be to build or buy. When going down the path of building it is critical to understand the true cost and impact to the business since it will require significant skill set changes and development as well as the adoption of software development standards for maintenance. When looking to buy automation platforms it is critical to look at the out of the box functionality and time to value for current use-cases and those on the horizon. Each of these decisions will have an impact across the people, products, and processes within the organization.

References

Ansible	https://www.ansible.com
Gluware	https://gluware.com/
GitHub	https://github.com/
Jinja2	http://jinja.pocoo.org/
Netmiko	https://github.com/ktbyers/netmiko
Paramiko	http://paramiko.org/
Pexpect	https://pexpect.readthedocs.io/en/stable/index.html
Python	https://www.python.org/
Python Package Index	https://pypi.org/
YAML	http://yaml.org/
Cisco Image source	https://www.cisco.com/c/dam/en/us/solutions/collateral/enterprisenetworks/software-defined-access/solution-overview-c22-739012.pdf

gluware®
Intelligent Network Automation

For more info visit ***gluware.com***