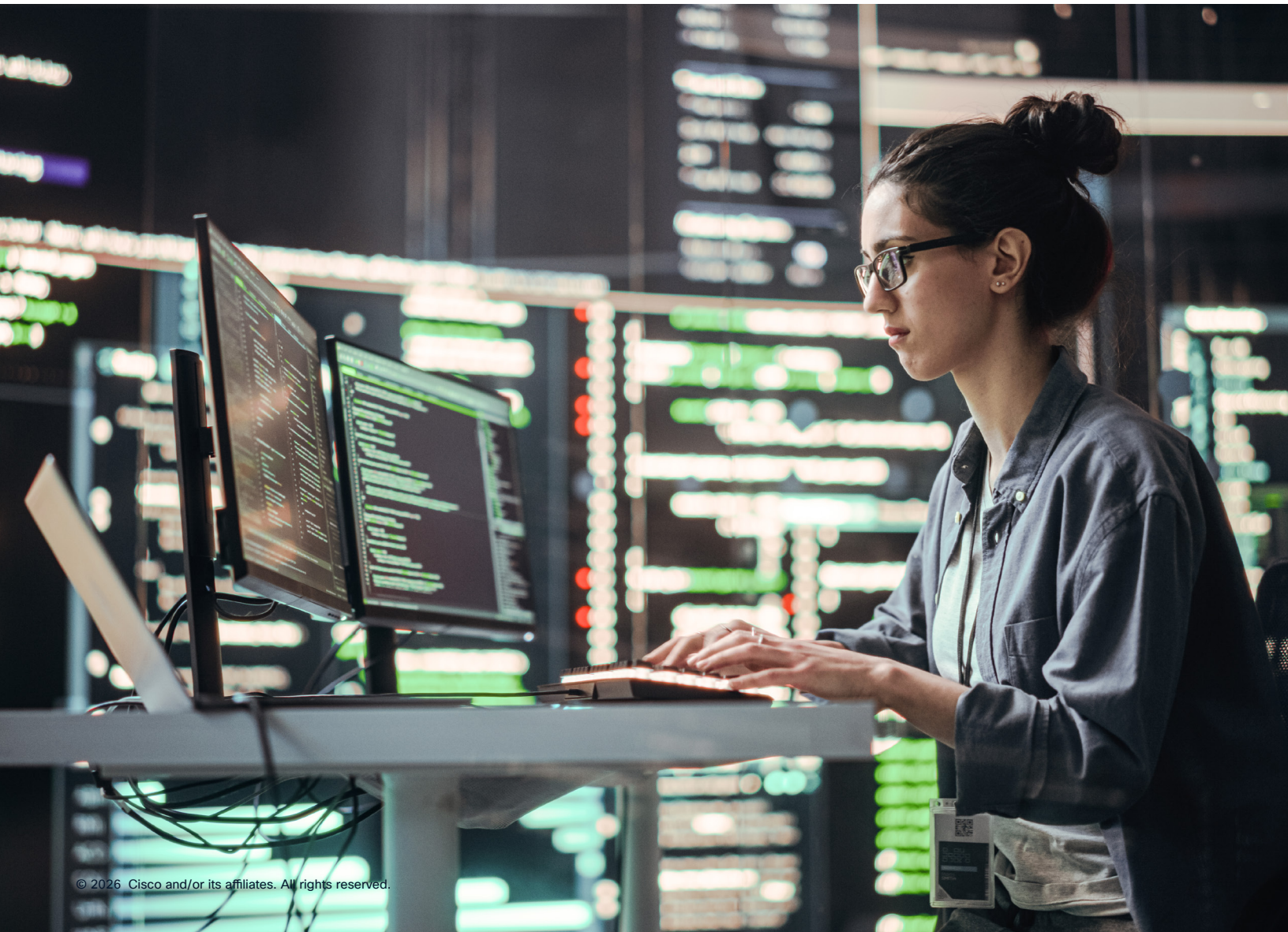
Automate Network Infrastructure Using Network as Code

Revolutionizes network management through data model– driven automation to simplify operations, ensure compliance, and increase change success rates.



© 2026 Cisco and/or its affiliates. All rights reserved.

Contents

© 2026 Cisco and/or its affiliates. All rights reserved.

Executive summary

In today's fast-paced digital environment, reducing downtime, increasing operational efficiency, and enabling rapid innovation are key to staying competitive. Adopting Network as Code for network configuration management offers transformative benefits by integrating automation, validation, and testing into operational workflows. This methodology leverages Infrastructure as Code (IaC) and DevOps principles, where version-controlled repositories are used to create a more efficient, resilient, and scalable infrastructure.

By shifting to Network as Code, organizations can achieve measurable results, such as reduced operational overhead, faster time to market, and improved network reliability.

This white paper explores the key components of the Cisco® Network as Code framework and offers practical guidance on the factors to consider when deploying it in real-world networks. Additionally, this paper includes real-life customer case studies that highlight the challenges addressed through Network as Code, the implementation strategies adopted, and the benefits of model-driven automation experienced by these customers.

By implementing a Network as Code approach, organizations can achieve greater operational efficiency, reduce risks, and adapt to change with confidence. This approach not only ensures a more stable and predictable infrastructure but also accelerates time to value, fosters collaboration across teams, and empowers organizations to innovate at speed.

Introduction

Network infrastructures are complex environments comprising multiple technologies, often managed in different ways and potentially under the responsibility of various teams. Historically these infrastructures have been configured manually, a process which has several challenges—especially with the network infrastructure and design growing organically over time:

- **Limits Operational Efficiency:** Manual processes are inherently time-intensive and require significant human effort for tasks such as configuration, deployment, and testing. These processes do not scale well with increasing complexity or growing infrastructure demands, leading to inefficiencies in resource utilization.
- **Increased Risk of Human Errors:** Human involvement in repetitive and detailed tasks, such as network configuration, is prone to errors including typos, misconfigurations, or omissions. These errors can lead to inconsistent deliverables, which may impact the reliability and performance of network infrastructure.
- **Inconsistent Configurations:** Variations in how tasks are performed manually can lead to discrepancies in network configurations across environments. Inconsistent configurations may cause operational issues, such as compatibility problems, security vulnerabilities, or suboptimal performance.
- **Cumbersome and Error-Prone Processes:** Manual delivery methods for planning, deployment, and testing, are labor-intensive and prone to oversight. These processes slow down project timelines and reduce the overall agility of the organization, making it difficult to adapt to changing requirements.

© 2026 Cisco and/or its affiliates. All rights reserved.

- **Inflexibility to Adapt to Evolving Requirements:** Throughout the lifecycle of a network infrastructure, requirements may change due to factors like capacity adjustments, new capabilities, or updated security needs. Manual implementation of these changes requires updates to multiple artifacts, which is time-consuming and labor-intensive. This model becomes economically unviable as networks grow in complexity, especially when rapid adaptation is essential.

One of the ways to address the challenges associated with managing network infrastructures manually is to look at how to automate the otherwise manual tasks. Automation not only enables configuration changes to be performed faster, but it also improves configuration consistency, as, for example, two changes with the same input parameters will be performed in the same way.

Automation as an Enabler for AI-Driven Networks

AI systems, such as AI Agents or AI-powered Operations (AIOps), require structured, machine-readable data to function effectively. Manual processes often generate unstructured or inconsistent data, making it difficult for AI systems to process and analyze the information.

Autonomous networks and AIOps rely on automation, real-time data, and consistent configurations to enable intelligent decision-making and self-optimization. Manual processes hinder the ability to implement these capabilities, as they lack the speed, accuracy, and standardization needed for advanced AI-driven operations.

Evolution of Network Automation

Automation of network equipment has existed in some form since the early days of the internet. Engineers would write code that would interact directly with the Command Line Interface (CLI) of these devices in a way that was directly representative of the human experience. This method of automation was cumbersome and prone to error since it would require that engineers keep updating automation every time a change was made to the CLI.

Various programming languages and methods were created to “crawl” CLI interfaces until the adoption of more programmatic interfaces had been developed. These programmatic interfaces interact with network equipment utilizing Application Programming Interfaces (APIs) that created a more “computer-friendly” approach, which allows the usage of data structures designed for computer consumption instead of human consumption.

These new data structures were tied with APIs in the network devices to allow for structured computer code interaction, in methods that facilitated both changing the configurations and reading telemetry data. Over time, the network has evolved to allow for both humans and computers to interact with network equipment in an efficient manner.

At the same time, the evolution of cloud computing brought in an era of new “infrastructure automation” to manage large-scale cloud environments. Developers would build virtual infrastructure to manage cloud applications programmatically, and the terminology “Infrastructure as Code” was born [1].

© 2026 Cisco and/or its affiliates. All rights reserved.

The shift to network automation based on Infrastructure as Code principles has led to Cisco developing Network as Code, a framework of capabilities that provides Infrastructure as Code methodology to manage network infrastructures. Network as Code has already been adopted by more than 100 customers and supports 11+ architectures, including Cisco Application Centric Infrastructure (Cisco ACI®), Virtual Extensible LAN (VXLAN) Ethernet Virtual Private Network (EVPN), Cisco Software-Defined WAN (Cisco SD-WAN), Cisco Meraki™, Cisco Identity Services Engine (Cisco ISE), Cisco Catalyst® Center, and more.

The framework consists of multiple building blocks, most of which are provided as publicly available and do not require any license or support contract to be used. The publicly available components include the automation code itself, tooling that can be used for pre- and post-change validations, etc.

This white paper goes into detail about how Network as Code works and how, through data model-driven automation, it can speed up the deployment of automation. The tools and methodology are applicable for all technologies where a data model is available, although most of the examples and scenarios are explained in the context of the data center.

Cisco Customer Experience (CX) provides a comprehensive suite of services designed to help and assist customers deploying Network as Code in their environment.

Customers who adopt a Network as Code framework often experience measurable improvements in key performance areas such as:

- **98%+ Configuration Change Success:** Automation reduces human intervention, minimizing mistakes and the operational overhead required to fix them.
- **5x Faster Deployment Times:** Streamlined workflows and pre-tested updates accelerate time to market for new services and capabilities.
- **Improved Reliability and Uptime:** Incremental and validated changes reduce downtime, ensuring critical business operations remain uninterrupted.

© 2026 Cisco and/or its affiliates. All rights reserved.

Cisco Network as code

Cisco Network as Code treats network infrastructure and services as code, enabling automation, version control, and Continuous Integration/Continuous Deployment/Continuous Testing (CI/CD/CT) practices, to manage and provision networks programmatically. This methodology applies DevOps principles to replace traditional manual network configuration with code-based automation, improving reliability, scalability, and efficiency.

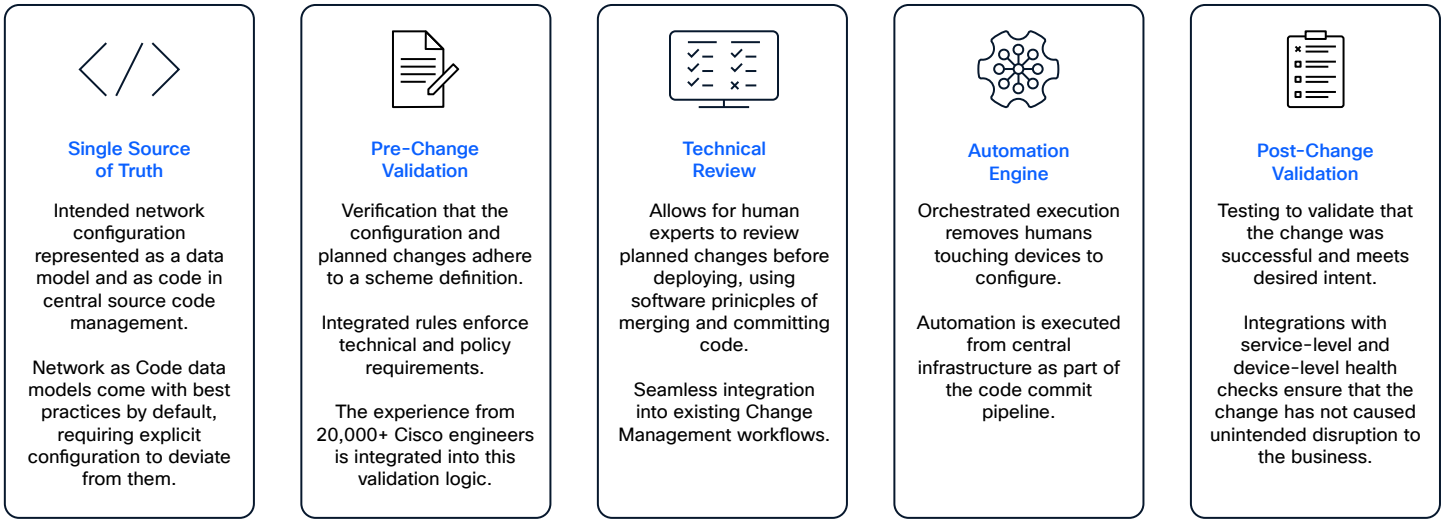


Figure 1. Components of Cisco Network as Code

Based on the same principles as Infrastructure as Code, but enhanced by Cisco, Network as Code provides network operators multiple benefits over traditional network provisioning and change. Some of these benefits include:

- **Single Source of Truth:** Intended network configuration represented in a data model and as code in Source Code Management (SCM) systems. This ensures that everyone relies on the same, consistent data for configuration decisions [2].
- **Version Control:** Provides all the known benefits of source control, such as parallel development, branching, conflict resolution, change history, rollback capability, and auditing. Storing network configurations in version control systems (e.g.,Git), enables teams to track changes, collaborate effectively, and roll back to previous configurations if needed.
- **Pre-Change Validation:** Multiple levels of validation can be applied. Syntax validation is used to verify data model input, while semantic validation is used to enforce compliance with network policies, standards, and best practices. An example of this could be to ensure uniqueness of VLAN IDs or to prevent overlapping IP subnets.

  Where available, we leverage any built-in pre-change capabilities of existing software/network controllers. An example is to integrate with Cisco Nexus® Dashboard Insights (NDI) and leverage its pre-change compliance capabilities in addition to the ones provided by Network as Code.
- **Post-Change Validation:** After a network change, integrations with service-level and device-level health checks can ensure that the change has not caused unintended disruption.

© 2026 Cisco and/or its affiliates. All rights reserved.

Network as Code also abstracts the definition of the intended network configuration by creating a simplified, declarative, human-readable data model, expressed in Yet Another Markup Language (YAML) files, that represents each network architecture. These data models are constructed by network professionals who understand the complete solution architecture, and they can then be utilized to simplify network provisioning. This simplification enables network engineers to focus on network intent (instead of code), alleviating the need to understand scripting and specific programming languages.

A simple ACI [3] example data model is listed below.

```
---
apic:
  tenants:
    - name: Tenant1
      vrfs:
        - name: VRF1
        - name: VRF2
```

Figure 2. Simple ACI as Code Example

The schema underpinning this network data model ensures that each parameter in the intent state definition (YAML content) is properly validated. This schema makes it possible to catch simple errors before automation is executed against devices in the network.

All the above concepts are then integrated into an automated CI/CD pipeline. The configuration of the pipeline is flexible and allows the user various ways to customize their environment. Some examples include:

- Code linting validating the syntax and formatting
- Defining what “stages” are executed for each branch or network change
- Integration with test automation platforms, existing CI/CD tools, third-party tools, and external systems
- Autogenerate test reporting

© 2026 Cisco and/or its affiliates. All rights reserved.

The figure below illustrates a sample CI/CD pipeline for ACI as Code.

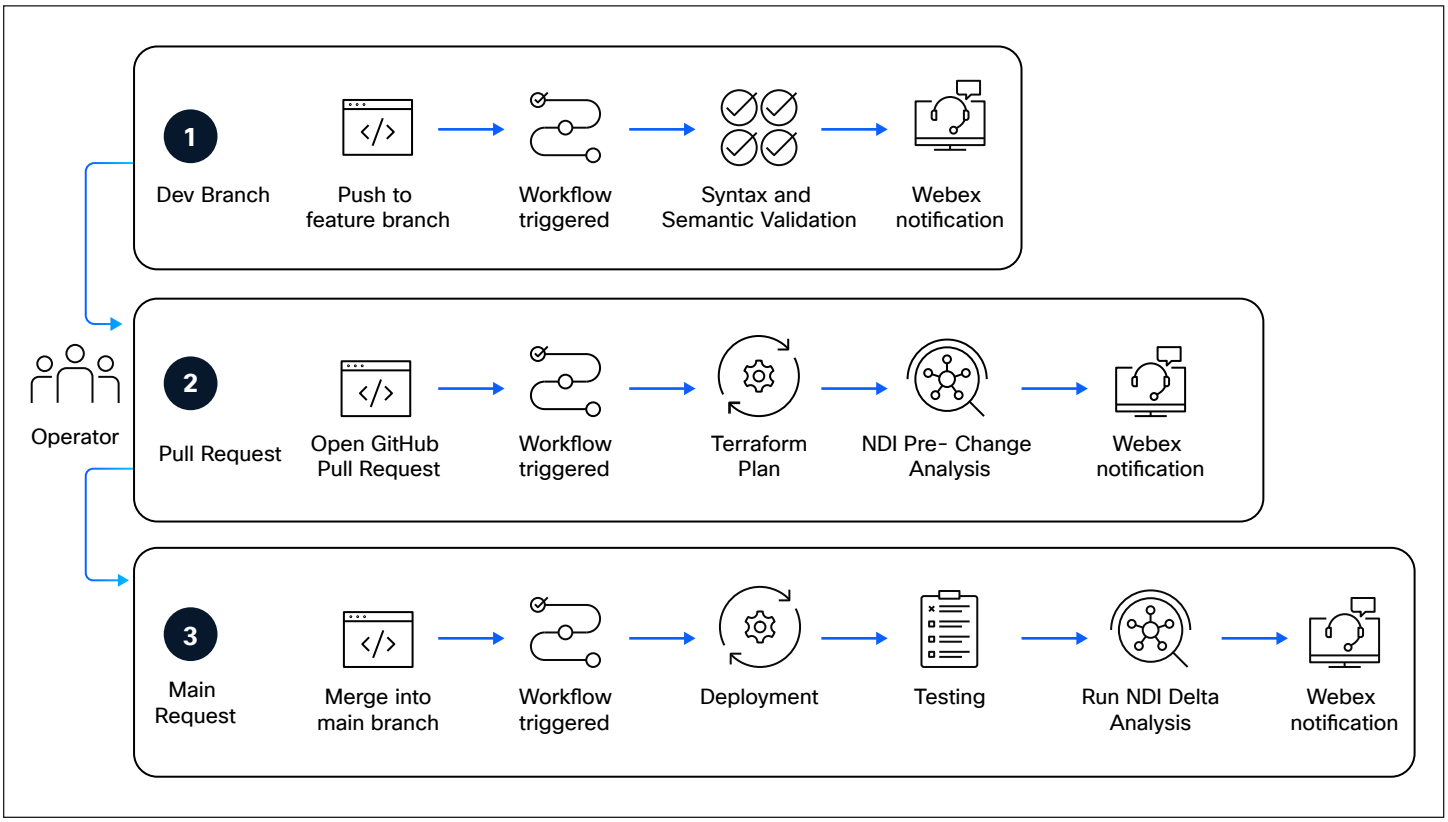


Figure 3. Sample ACI as Code CI/CD Pipeline

By leveraging a CI/CD pipeline, network changes now benefit from repeatable automation, consistent validation processes, faster deployment cycles, and enhanced reliability.

Integration with workflow orchestrators

Managing network infrastructure using Network as Code often involves interaction with a variety of systems and tools within the broader IT ecosystem. These may include service catalogs or portals where end users request network services, as well as workflow orchestrators that coordinate complex processes. To enable such integrations, Network as Code provides a standardized mechanism to expose its automation capabilities through APIs.

At the core of this integration is the NAC-API (Network as Code API), which acts as the central interface between automation frameworks and network infrastructure. By offering well-defined and standardized APIs, the NAC-API enables developers to programmatically configure network devices and services. This API-driven approach ensures seamless integration with third-party systems, fostering an interconnected ecosystem of tools and facilitating end-to-end automation workflows with consistency and scalability.

© 2026 Cisco and/or its affiliates. All rights reserved.

Using AI to support workforce transformation

The shift from traditional, manual network operations to an automation-driven approach is not just a technological change—it's a journey of workforce transformation. To aid in this transition, Network as Code integrates an AI Assistant designed to act as an intelligent collaborator for network engineers. By harnessing the power of generative AI and natural language processing, the AI Assistant provides contextual insights and real-time recommendations, empowering engineers to tackle complex challenges with greater confidence and efficiency.

From understanding your configuration and responding to your engineers, generating configurations based on the intent for your desired configuration, or assessing and providing recommendations for issues during deployment, the AI Assistant simplifies the management of network configurations and operations. By bridging the gap between human expertise and automation, the AI Assistant accelerates the adoption of Network as Code while ensuring a smoother and more productive transition for the workforce.

More about Network as Code

The Network as Code solutions are constantly being enhanced to add new features and functionality, as well as expanded to cover additional architectures.

If you want to keep up to date with the latest features and functionality, please visit https://netascode.cisco.com/. This website not only contains always up-to-date documentation about the architectures supported and their corresponding data models, but it also contains a wealth of information about how to use and deploy open-source solutions.

Architecture components

The architecture of Network as Code is built on a foundation of modular and interoperable components, each playing a critical role in enabling seamless automation and orchestration. These components work together to abstract network complexity, streamline operations, and ensure scalability. This chapter explores the key building blocks—ranging from data models and APIs to workflow orchestrators and AI-powered assistants—that collectively drive the transformation to an automated, programmable network.

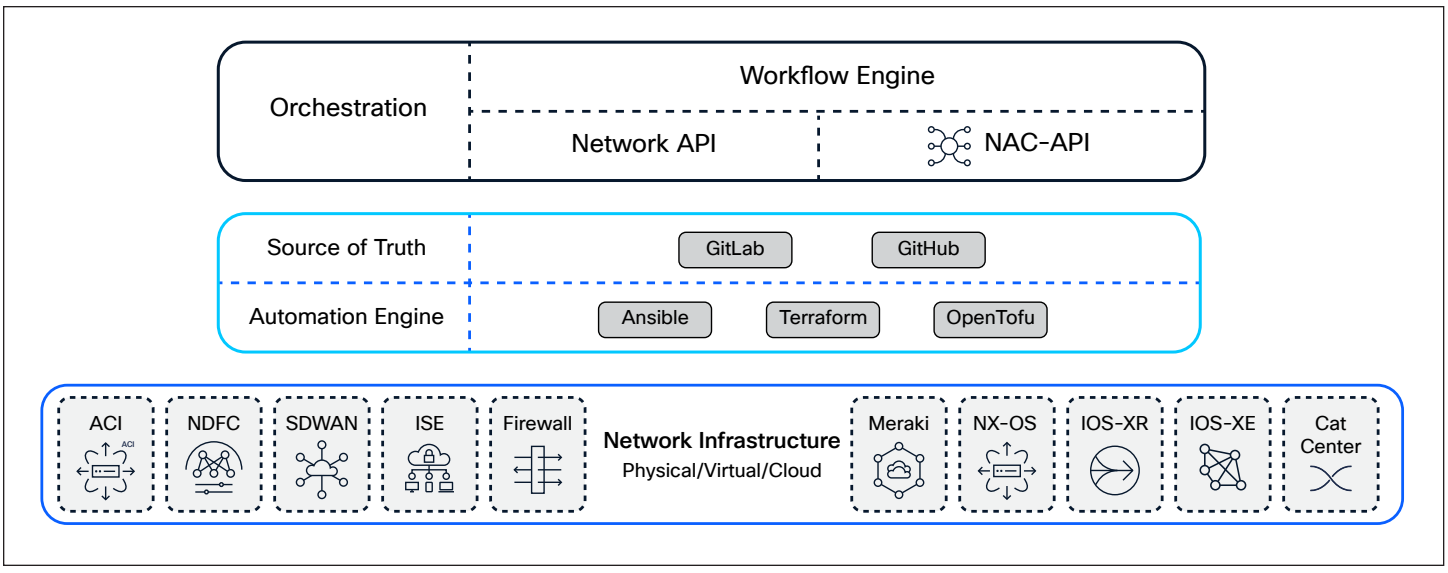


Figure 4. Network as Code Architecture Components

© 2026 Cisco and/or its affiliates. All rights reserved.

Using git as source of truth

One of the common challenges in Day-2 Network Operations is the lack of a Single Source of Truth (SSOT) for network configuration state. This can lead to inconsistencies, confusion, and inefficiencies in managing network resources. Often, network teams rely on multiple sources of truth, such as isolated automation tools, Configuration Management Databases (CMDBs), IP Address Management (IPAM), spreadsheets, and manual documentation, which can become outdated, inconsistent, or inaccurate [2].

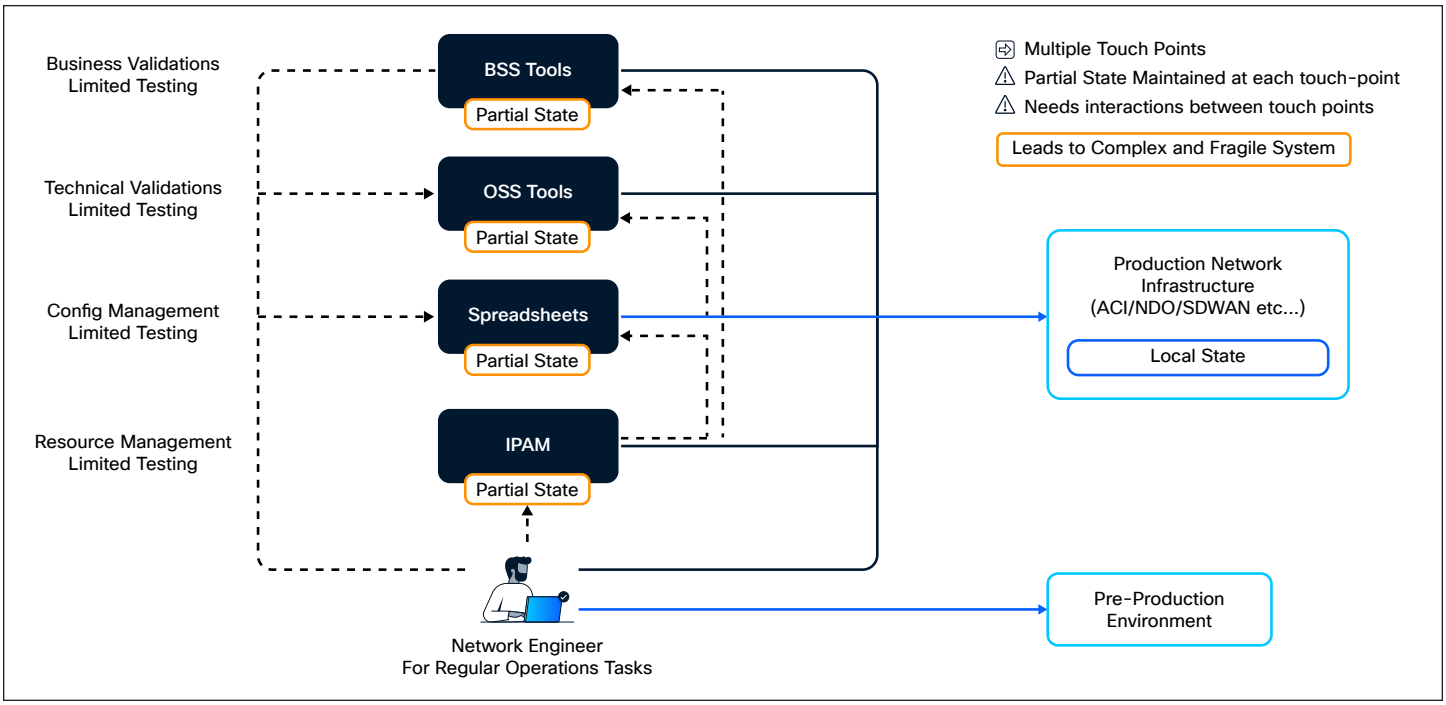


Figure 5. Multiple States with Complex Interaction

It is difficult to maintain a consistent and accurate view of the network state when relying on multiple sources. Imagine a scenario when a network engineer must provision a new service, and the dependency information is scattered across multiple systems. This can lead to delays, errors, and frustration. Thus, having a Single Source of Truth is crucial for efficient and effective network operations.

© 2026 Cisco and/or its affiliates. All rights reserved.

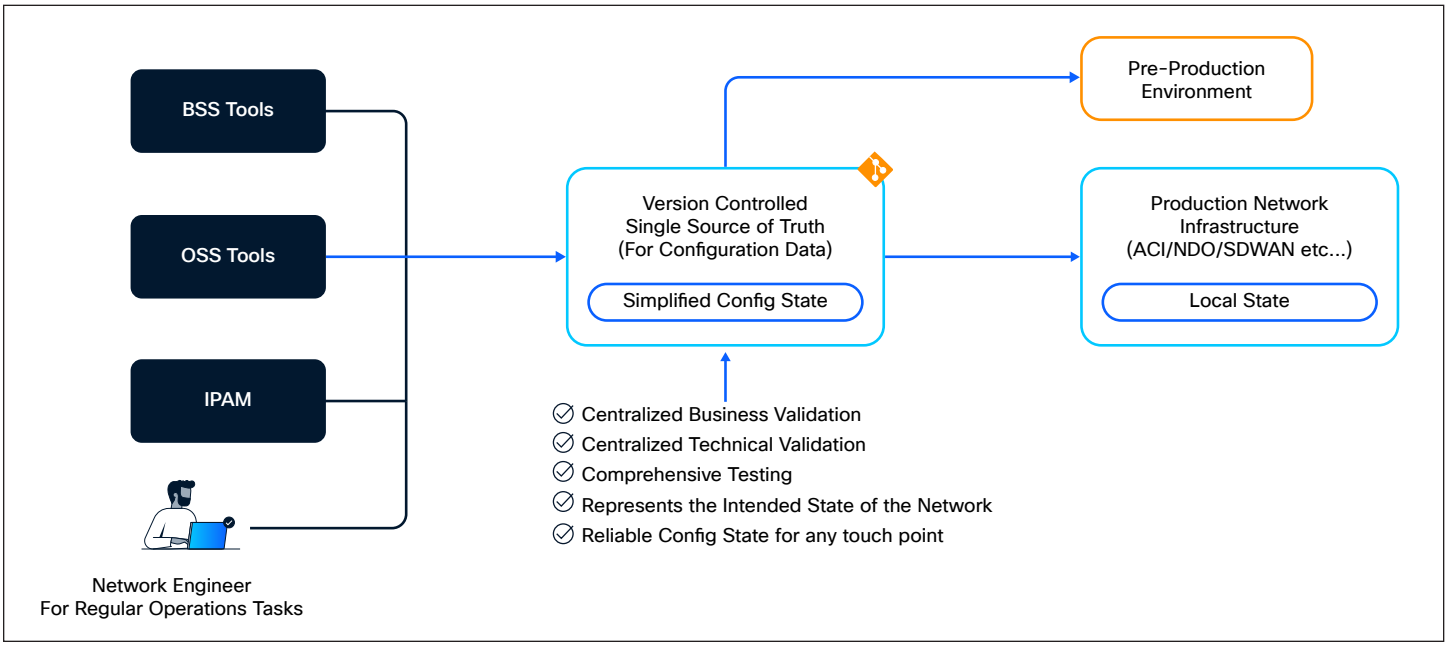


Figure 6. Single Source of Truth for Simplified and Efficient Operations

With Network as Code (NaC), the Single Source of Truth (SSOT) is represented by a version-controlled repository in Git that contains the desired state of the network configuration. This desired state can thus be used to drive automation tools, such as Ansible, Terraform, and OpenTofu, to ensure that the actual state of the network matches the desired state. At the same time, the SSOT can be used for automated validation, testing, and compliance checks, ensuring that the network configuration is always in the expected state.

Here are the three key benefits of having a Single Source of Truth (SSOT) in network operations:

- **Configuration Consistency:** Adopting SSOT makes it easier to maintain a consistent and accurate view of the network configuration. This is done by ensuring that the data in SSOT conforms to the defined schema of target systems, validated for correctness of data, covering both technical and business aspects. This reduces the risk of misconfigurations and ensures that the network operates as intended.
- **Operational Efficiency:** Once an SSOT is established, changes to the intended state of the network can be managed more efficiently, using GitOps practices. This allows for incorporating multiple validations, pre-production checks, and an expert review process before changes are applied to the SSOT, which will then reflect in the production environment. The SSOT can also be used to automate routine tasks, such as configuration backups, compliance checks, and reporting, freeing up network engineers to focus on more strategic tasks.
- **Collaboration and Transparency:** An SSOT fosters collaboration among network teams, as it provides a common reference point for all stakeholders. This transparency helps in reducing misunderstandings and miscommunications, as everyone is working from the same set of data. It also enables better tracking of changes, accountability, and auditability, which are essential for compliance and governance.

© 2026 Cisco and/or its affiliates. All rights reserved.

Network as Code Data Models

A data model is a representation of the entities structure, relationships, and attributes of the data within a system or domain. It provides a conceptual or logical framework for organizing and describing data elements and their interactions. In the case of network data models:

- **Entities** include sites, buildings, network devices, network device interfaces, virtual overlay networks, endpoints attached to the network, etc.
- **Relationships** define association of entities. For example, a switch belongs to a building, and multiple access points are attached to a switch. These relationships help to define the hierarchy of the network entities.
- **Attributes** are mainly proprieties or the network entities, including configuration parameters such as Virtual Routing and Forwarding (VRF) names, IP addresses, network policies, and more.

Data models help ensure data consistency, integrity, and accuracy by settings rules and constraints for data representation and relationships. In addition, structuring data according to a data model enables efficient querying and retrieval of information from a Source of Truth, as well as support for data manipulation operations. Data models also facilitate communication and understanding among stakeholders in the organization [2].

IIn the context of network automation, a well-designed data model is crucial for the success of automation projects. The example in figure below describes the Bidirectional Forwarding Detection (BFD) switch policy for the Cisco Application Policy Infrastructure Controller (APIC) data model. You can find definitions of all Cisco Network as Code data models at https://netascode.cisco.com/.

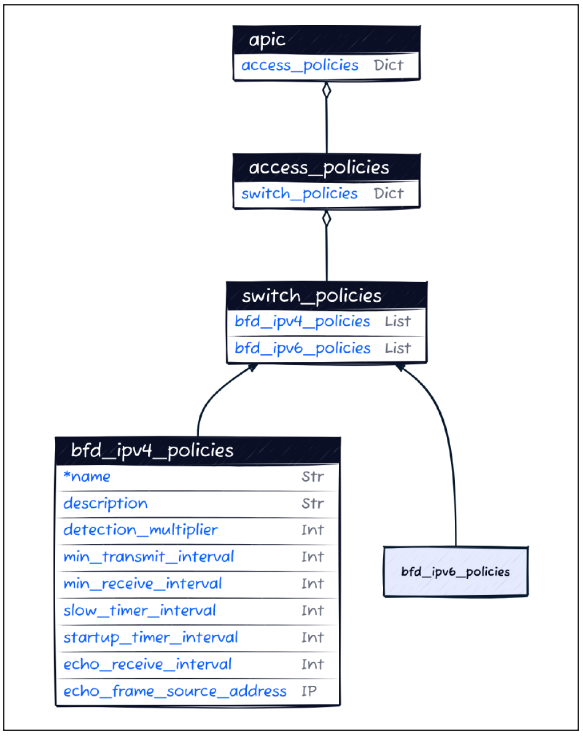


Figure 7. Data Model for a BFD Switch Policy in ACI

© 2026 Cisco and/or its affiliates. All rights reserved.

Data models can be either vendor-specific (such as Cisco Network as Code data models) or vendor-agnostic (like OpenConfig [4]). IIn Cisco's experience, most enterprise organizations rely on vendor-specific data models for implementation of automation.

In the case of Cisco Network as Code, YAML format is used. When working with YAML files, the Cisco Network as Code uses Yamale Schema Validator [5] —a powerful tool that enables the definition of schemas using a YAML-based syntax. Yamale allows specifying rules and constraints for the structure and content of YAML data, checking whether the file adheres to the defined schema and providing validation results. The table below describes an IPv4 BFD section APIC BFD switch policy schema.

Table 1. Section of the ACI as Code Data Model's Schema

Name	Type	Constraint	Mandatory	Default Value
name	String	Regex: ^[a-zA-Z0-9_.:-]{1, 64}$	Yes	
alias	String	Regex: ^[a-zA-Z0-9\\!#$%()*,-./:;@ _{\|}~?&+]{1, 128}$	No	
data_plane_learning	Boolean	true, false	No	true
enforcement_direction	Choice	ingress, egress	No	egress
enforcement_preference	Choice	enforced, unenforced	No	enforced
bgp	Class	[bgp]	No	
dns_labels	List	String[Regex: ^[a-zA-Z0-9_.:-]{1, 64}$]	No	

Cisco Network as Code differentiates between three types of data models:

- **Controller-Centric Data Models:** These data models are built around APIs exposed by controllers, such as APIC [3], Catalyst SD-WAN Manager [6], or Meraki Dashboard [7]. The model is typically closely aligned with the API structure and is used to configure the controller.
- **Solution-Centric Data Models:** These data models are built around a specific solution, for example VXLAN EVPN, which can be deployed to multiple platforms and/or controllers. VXLAN EVPN is one such solution, which can be deployed to Catalyst switches, Nexus switches, with or without controllers.
- **Device-Centric Data Models:** These data models are built around the configuration of individual network devices, such as Catalyst switches, routers, or Nexus switches. The model is typically aligned with the CLI device configuration structure and is used to configure the device.

The choice of data model often depends on the specific network infrastructure being automated. In cases where the network includes a tightly integrated controller, such as the APIC or Catalyst SD-WAN Manager, the appropriate data model is typically straightforward and aligned with the solution. However, in scenarios where the controller is less tightly coupled with the network solution, the decision becomes less clear, prompting consideration of whether to adopt a solution-centric or device-centric data model.

© 2026 Cisco and/or its affiliates. All rights reserved.

Automation engine

Automation Engine is the core component of the Network as Code framework. It is responsible for applying configurations to the infrastructure in a consistent, efficient, and error-free way.

The automation engine is responsible for implementing the configuration abstraction introduced by the data model. In other words, the automation engine is responsible for translating the desired configuration into controller- or device-level configurations that can be pushed to the infrastructure.

When it comes to the implementation of the automation engine, there are several fundamentally different approaches that can be chosen:

- Imperative/procedural vs. declarative
- Stateful vs stateless

The imperative/procedural approach specifies the sequence of steps required to achieve a certain outcome—i.e., do task A, do task B, verify task C, and do task D, in that exact order. Such an approach is not ideal for configuration tasks, as the described procedure often assumes a given state of the infrastructure when starting the workflow. As an example, how would the workflow detect that task A has already been completed and therefore can be skipped—especially if the operation in task A is not idempotent, meaning that if it is run again for the same target, it will not create the same results. The imperative/procedural approach to automation is often required for use cases such as software upgrades, as these typically need a specific sequence of tasks to be executed in a predetermined sequence of steps and validations.

The declarative approach to automation instead focuses on describing the desired state for the infrastructure and leaves it up to the automation system to derive the changes and sequence of actions that need to occur for the infrastructure to have the desired state. A declarative implementation is ideally suited for managing infrastructure configuration.

With a stateless approach to automation, the automation solution does not have any knowledge about what it has done previously. Good candidates for automation tasks using a stateless approach are functions such as software upgrades, where only an indication of success or failure after the execution is needed.

A stateful approach to automation, on the other hand, stores the state of the infrastructure between execution runs, allowing for detection of configuration changes performed outside of the automation framework—also known as configuration drift. Stateful implementations of automation are typically seen when the automation solution is used to manage the configuration of the infrastructure.

© 2026 Cisco and/or its affiliates. All rights reserved.

For the implementation of the automation system, multiple tools and solutions are typically used. Popular choices for the configuration management elements include:

- RedHat Ansible
- HashiCorp Terraform
- OpenTofu

Ansible [8] is an automation tool from RedHat that simplifies the management and configuration of a wide range of systems. It uses YAML to define the order in which the tasks should be executed on the target devices to achieve the desired configuration. Ansible does not require any agents to be installed on the target systems. It is based on an imperative/procedural approach to automation in the sense that you define a sequence of tasks that are executed in the specified sequence. It is possible to achieve declarative behavior with Ansible; however, this requires special care and considerations when writing the Ansible Playbooks and comes with some additional complexity—especially if state is required between executions to achieve the declarative behavior, as Ansible is stateless by nature.

Terraform [9] is an infrastructure automation tool from HashiCorp that enables a wide range of infrastructure resources to be configured based on a declarative intent definition. This definition is specified using a human-readable format called HashiCorp Configuration Language (HCL). Once the desired state is specified, Terraform will automatically determine the configuration additions, updates, and removals required in the infrastructure to ensure that it matches the defined intent. Terraform keeps state between executions, and this state information is used when determining the actions required to achieve the defined intent.

OpenTofu [10] is an open-source alternative to Terraform and can serve as a drop-in replacement allowing existing workflows and configurations to be used across OpenTofu and Terraform.

Ansible and Terraform each have their strengths and weaknesses, which means there often is not one tool that fits all automation use cases. Ansible is often used to implement use cases such as software upgrades of network devices or to manage the lifecycle of software packages on server workloads, while Terraform is used to manage the lifecycle of infrastructure configurations including networks/VLANs, deployment of VMs, etc.

In a Network as Code framework, technologies are mainly automated with Terraform, with some exceptions that primarily use Ansible. Each solution working with Terraform is based on a Terraform provider. The provider is a plugin that allows Terraform to interact with various network infrastructure components. For example, the ACI provider can communicate only with APIC, and the SD-WAN provider can communicate only with vManage. It abstracts the underlying APIs, allowing users to define and manage network configurations in a declarative manner using HashiCorp Configuration Language (HCL).

Using the Network as Code framework, the automation engine, i.e. the code written in HCL that abstracts the data model, is already created and published by Cisco in a ready-to-use form as a Terraform module.

© 2026 Cisco and/or its affiliates. All rights reserved.

A Terraform module is a reusable, pre-defined configuration that contains multiple resources and their dependencies. As an example, a Terraform resource represents a single API object, in the case of ACI a single Managed Object (MO), while a Terraform module may consist of multiple resources, such as a branch of MOs in case of ACI, such as full Endpoint Group configuration.

With Network as Code, a single module can manage the entire configuration of a solution such as ACI or SD-WAN, and inside the module there are logical separations, which enables users to manage only pieces of configuration.

In the case of the ACI module there are six configuration sections that can be selectively enabled or disabled using module flags.

- **fabric_policies: C**onfigurations applied at the fabric level (e.g., fabric Border Gateway Protocol (BGP) route reflectors).
- **access_policies:** Configurations applied to external facing (downlink) interfaces (e.g., VLAN pools).
- **pod_policies:** Configurations applied at the pod level (e.g., TEP pool addresses).
- **node_policies:** Configurations applied at the node level (e.g., out-of-band [OOB] node management address).
- **interface_policies:** Configurations applied at the interface level (e.g., assigning interface policy groups to physical ports).
- **tenants:** Configurations applied at the tenant level (e.g., VRFs and Bridge Domains).

The Network as Code Terraform modules ship with default values for certain objects that codify best practices. These values are documented in the Data Model and embedded in a module. From a user perspective, a single file `defaults.nac.yaml` can be used to define custom definitions in a central location. This file overwrites any default values that come with the main modules.

This file is typically customized to reflect the specific customer requirements and reduces the overall size of input files as optional parameters with a default value that can be omitted. As an example, some customers prefer to append suffixes to object names to comply with a naming convention. Such suffixes can be defined once in `defaults.nac.yaml` and then consistently appended to all objects of a specific type including its references. If an ACI fabric is expected to mainly work as a Layer-2 fabric, the Bridge Domains settings corresponding to the flooding and routing mechanism can be included in the defaults file and wouldn't have to be explicitly configured with each new Bridge Domain in the input file.

Solutions working with Ansible, such as the Cisco Nexus Dashboard Fabric Controller (NDFC) [11] fabrics, rely on Ansible collections. Ansible collections are a standardized way to organize and package Ansible content, including roles, modules, utilities, plugins etc. The Collection for NDFC provides pre-built modules to manage various network functions, such as creating fabrics, adding switches, and configuration interfaces and networks.

© 2026 Cisco and/or its affiliates. All rights reserved.

Automated Testing to provide Change Assurance

In the dynamic and increasingly complex landscape of modern networks, manual configuration and change management are significant inhibitors to agility and reliability. Statistics indicate that a substantial majority of network problems, often exceeding 80%, stem from improper configurations and issues within change management processes. These errors lead to costly downtime, security vulnerabilities, and a general erosion of operational efficiency. To mitigate these pervasive challenges, the Cisco Network as Code (NaC) framework introduces a robust, multi-layered automated testing and validation strategy, fundamentally transforming change management into a predictable and reliable process.

The core principle behind NaC's approach to automated testing is "shift-left" validation. This methodology advocates for integrating testing and quality assurance as early as possible in the development and deployment lifecycle, and remediating potential issues before they can impact the live network. By doing so, NaC drastically reduces the risk associated with network changes, accelerates deployment cycles, and ensures the continuous integrity of the network infrastructure. This comprehensive testing framework is built upon two primary layers of validation and testing:

- Pre-Change Data Model Validation
- Post-Change Configuration, Operational State, and Health Verification

Data Model Validation with nac-validate

The journey of any network change within the NaC model begins with the declarative data model, defined in human-readable YAML files. This data model represents the desired state of the network. The `nac-validate tool` [12] serves as a critical component to ensure successful network changes, rigorously ensuring the integrity and correctness of this source of truth before any configuration is pushed to the live network. In other words, nac-validate is leveraged pre-change.

nac-validate performs a multi-stage validation process:

- **Format Validation:** At the most basic level, `nac-validate` confirms that the desired state defined in YAML files is well-formed (e.g., valid YAML syntax), preventing fundamental structural errors.
- **Syntax Validation:** This step uses a predefined schema to enforce the structural rules of the data model. It verifies that all required keys are present, and data types are correct (e.g., ensuring a network segment ID is an integer within a valid range, or a device interface speed adheres to defined enumerations like "1G", "10G", "25G"). By catching these errors early, syntactic validation prevents a significant class of simple but potentially disruptive mistakes.

© 2026 Cisco and/or its affiliates. All rights reserved.

- **Semantic Validation:** Going beyond mere syntax, `nac-validate` performs a more sophisticated check to ensure the data is logically consistent and adheres to the intended network design. This prevents scenarios where, despite syntactically correct inputs, the combination describes a logically impossible or undesirable network state. Examples include:
  - **Referential Integrity:** Verifying that any reference to another network object (e.g., a port profile referencing a specific resource pool) is valid and that the referenced object exists within the data model.
  - **IP Address Management:** Checking for overlapping subnets within the same virtual network segment or ensuring that a gateway IP address is indeed part of the defined subnet.
  - **Cross-Object Logic:** Validating complex relationships between different parts of the data model, such as ensuring that a network segment defining a service relationship is not also consuming that same service, which could indicate a logical flaw in the network design.
- **Compliance Validation:** A powerful feature of `nac-validate` is its extensibility for custom policy enforcement. Organizations can codify and automatically enforce their own unique standards, security policies, and best practices. These custom rules, written in Python, can:
  - Enforce naming conventions for network objects (e.g., `<site _ code>-<function>-<unique _ id>`).
  - Prohibit risky configurations, such as overly permissive access policies or external network connections without appropriate security filtering.
  - Ensure regulatory compliance, for instance, by verifying that all network segments handling sensitive data have appropriate Quality of Service (QoS) policies or logging levels applied.
  - Prevent the allocation of resources that are reserved for specific infrastructure functions.

The primary benefit of this comprehensive pre-change validation is risk reduction. By identifying and rectifying errors, inconsistencies, and policy violations early in the CI/CD pipeline, `nac-validate` prevents faulty configurations from ever reaching the production environment. This significantly increases the reliability of changes and builds confidence in the automation process, embodying a true “shift-left” approach to network quality.

© 2026 Cisco and/or its affiliates. All rights reserved.

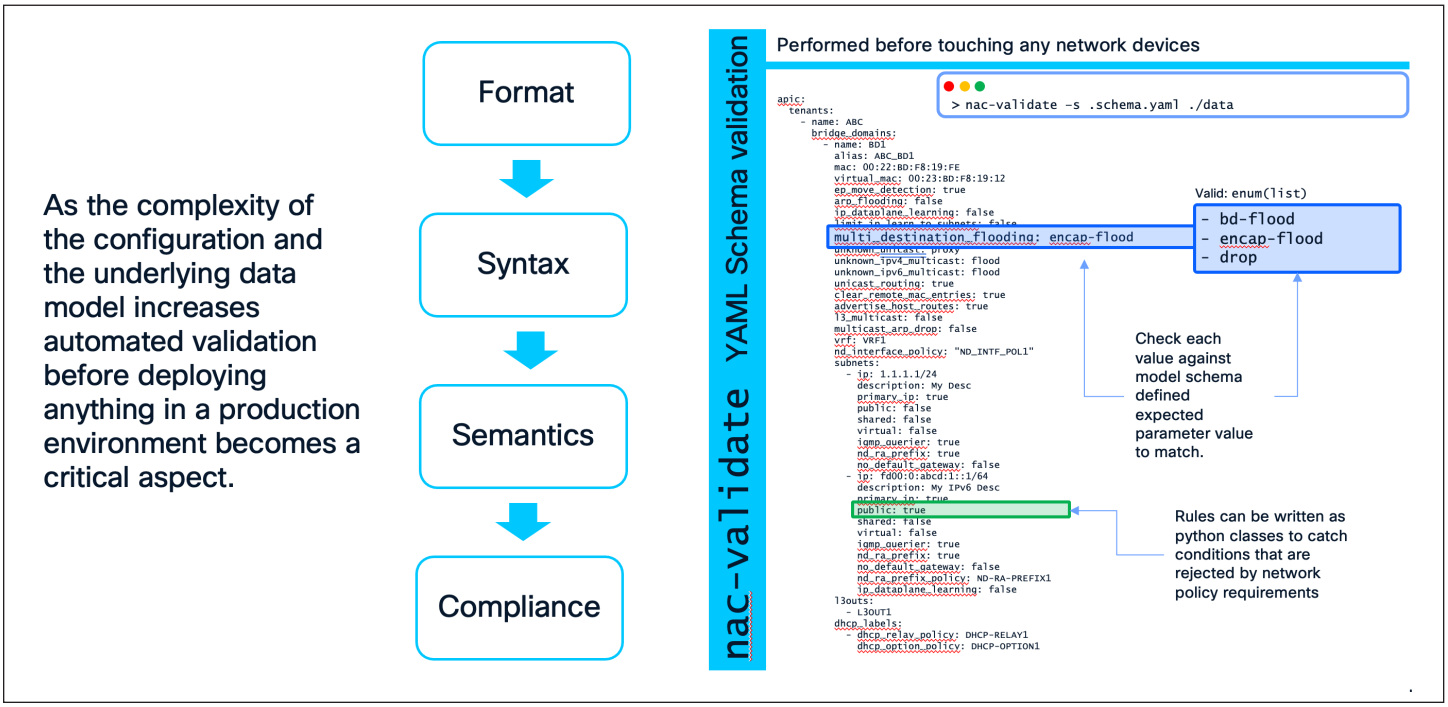


Figure 8. Pre-Change Data Model Validation

Configuration, Operational State, and Health Verification

Deploying a syntactically correct and logically sound configuration is a significant achievement, but it does not represent the end of the assurance process. The ultimate measure of a deployment's success lies in the actual operational state of the live network. The NaC testing framework extends its reach into the post-deployment phase with tests, executed by the `nac-test tool` [13].

These tests are designed to connect directly to the live network devices or controllers, query their current state, and validate that the network is not only configured as intended but is also behaving as expected.

- **Configuration Verification (Post-Change):** This component of `nac-test` directly compares the intended state (derived from the YAML data model) against the actual running configuration on the network devices or controllers. This ensures that the desired state has been successfully translated and applied to the infrastructure. It confirms that what was intended to be configured is indeed configured.
- **Operational State Tests:** These tests go beyond static configuration checks to validate the live, dynamic state and functionality of the network. They answer the critical question: "Is the network feature I just configured actually functioning as expected?" Examples include:
  - **Routing Protocols:** Verifying that dynamic routing protocol adjacencies are stable and that the expected route prefixes are being learned and advertised by peers.
  - **Tunnel Endpoints:** Confirming the operational status of overlay network tunnels and their reachability.

© 2026 Cisco and/or its affiliates. All rights reserved.

- **Health Tests:** These tests address a slightly different but equally vital question: “Did my change introduce any unintended side effects or degrade the overall health of the network?” They are designed to detect “brownfield” issues where the network might be “working” but is not “healthy.” Examples include:
  - **Faults and Alarms:** Checking for new system faults or critical alarms across network devices and controllers that may have appeared after a change was deployed, indicating underlying issues.
  - **Health Scores:** Monitoring the health metrics of critical network domains, virtual networks, or service groups to ensure they have not been negatively impacted or degraded.
  - **Interface Errors:** Monitoring critical interfaces for an increase in drops, errors, or Cyclic Redundancy Check (CRC) counters, which could indicate physical layer problems or misconfigurations.
  - **Resource Utilization:** Checking CPU, memory, and hardware resource (e.g., Ternary Content Addressable Memory [TCAM]) utilization on network devices to ensure a change has not inadvertently pushed a device beyond its recommended operational limits.

Please note that health tests may not be available for all architectures supported by the Network as Code framework.

The benefits of post-change validation are assurance and operational integrity. `nac-test` provides automated, undeniable proof that the change was successful and did not cause unintended side effects. This crucial step replaces manual, time-consuming, and error-prone post-change checkouts, enabling faster and more frequent deployments with confidence.

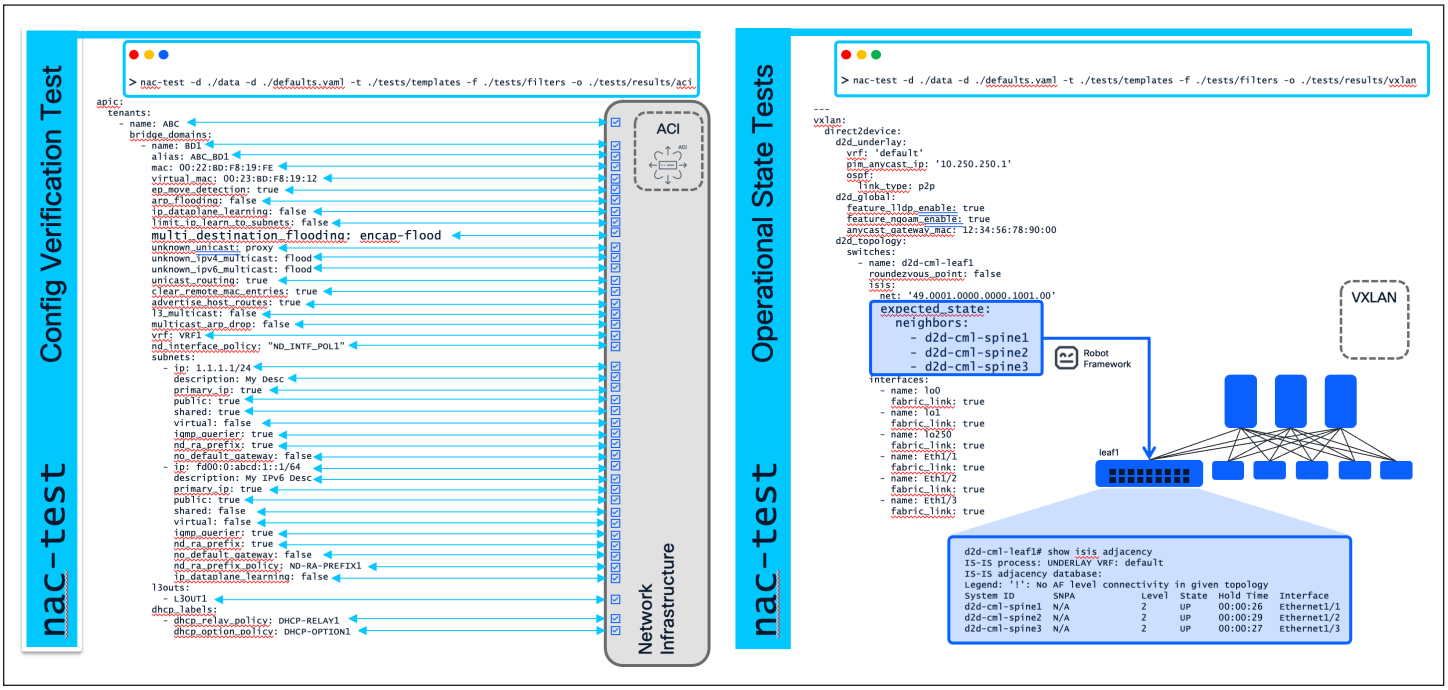


Figure 9. Post-Change Validation

© 2026 Cisco and/or its affiliates. All rights reserved.

The Benefits: Far More than the Sum of all the Parts

The adoption of this comprehensive, multi-layered automated testing and validation framework yields transformative benefits that extend far beyond simply preventing errors:

- **Massively Increased Velocity and Confidence:** When every change is automatically and rigorously tested at every level, the success rate of changes is significantly increased. Network operations teams can confidently push new features and updates, knowing that if a flaw exists, the automated pipeline will catch it early. This dramatically accelerates the pace of innovation and deployment.
- **Drastic Reduction in Mean Time to Recovery (MTTR):** In the infrequent event that an issue does occur, the failing test pinpoints the exact component and layer of the failure. A validation failure indicates an issue in the data model, a configuration test failure points to missing configurations, and an operational or health test failure highlights a specific live network component. This significantly reduces hours of manual troubleshooting and dramatically decreases the MTTR.
- **Living, Executable Documentation:** The entire suite of tests serves as a form of living, executable documentation. By examining the tests, an engineer can understand what standards are enforced, and what the expected operational state is. This documentation remains perpetually current because any deviation from reality will cause the tests to fail.
- **Democratization of Network Changes:** With a robust safety net in place, the ability to initiate and manage network changes can be extended beyond a small group of highly specialized network engineers.
  - Application teams, for example, could be empowered to manage their own connectivity requirements via pull requests to the data model, confident that the validation and testing pipeline will enforce all necessary guardrails and prevent unintended disruptions.
- **Enforced Standardization and Consistency:** The framework ensures that every corner of the network adheres to the same design patterns, naming conventions, and security policies. This leads to a more predictable, manageable, and inherently more secure network infrastructure, addressing the pain point of inconsistent deliverables.

By leveraging `nac-validate` for pre-change validation and `nac-test` for post-change verification, the Cisco Network as Code solution delivers an unparalleled end-to-end assurance model. This integrated approach transforms network management from a manual, error-prone endeavor into a predictable, reliable, and highly agile process, enabling organizations to achieve the consistency and rapid adaptation essential for modern digital infrastructures.

© 2026 Cisco and/or its affiliates. All rights reserved.

GitOps and CI/CD for network operations

The individual components of Network as Code, such as the data model-driven configuration intent stored in Git, pre- and post-deployment verification, etc., each provide great value. But the full value of the solution is realized when the execution of the individual components is automated. This is where Continuous Integration and Continuous Deployment (CI/CD) practices come into play, particularly when combined with GitOps principles.

Continuous Integration (CI), in the context of Network as Code, focuses on ensuring that any proposed changes to the network configuration are validated, tested, and integrated into the network in a controlled manner. This involves checking for data representation correctness, schema validation, semantic validation, best practices adherence, and compliance with organizational policies (if they are defined).

CI may also include additional testing against pre-production environments, such as staging or development networks, to ensure that the changes do not introduce any issues before they are applied to the production network. This process helps catch potential problems early, reducing the risk of downtime or misconfigurations in the live environment.

Continuous Deployment (CD) takes this a step further by automating the deployment of changes from the source of truth (Git) to the actual network devices or systems. This step will make use of an automation engine (Terraform, OpenTofu, or Ansible) to apply the changes defined in git to the production environment. After the changes are applied, it is essential to run comprehensive tests to ensure that the network is functioning as expected and that the changes have not introduced any regressions or issues.

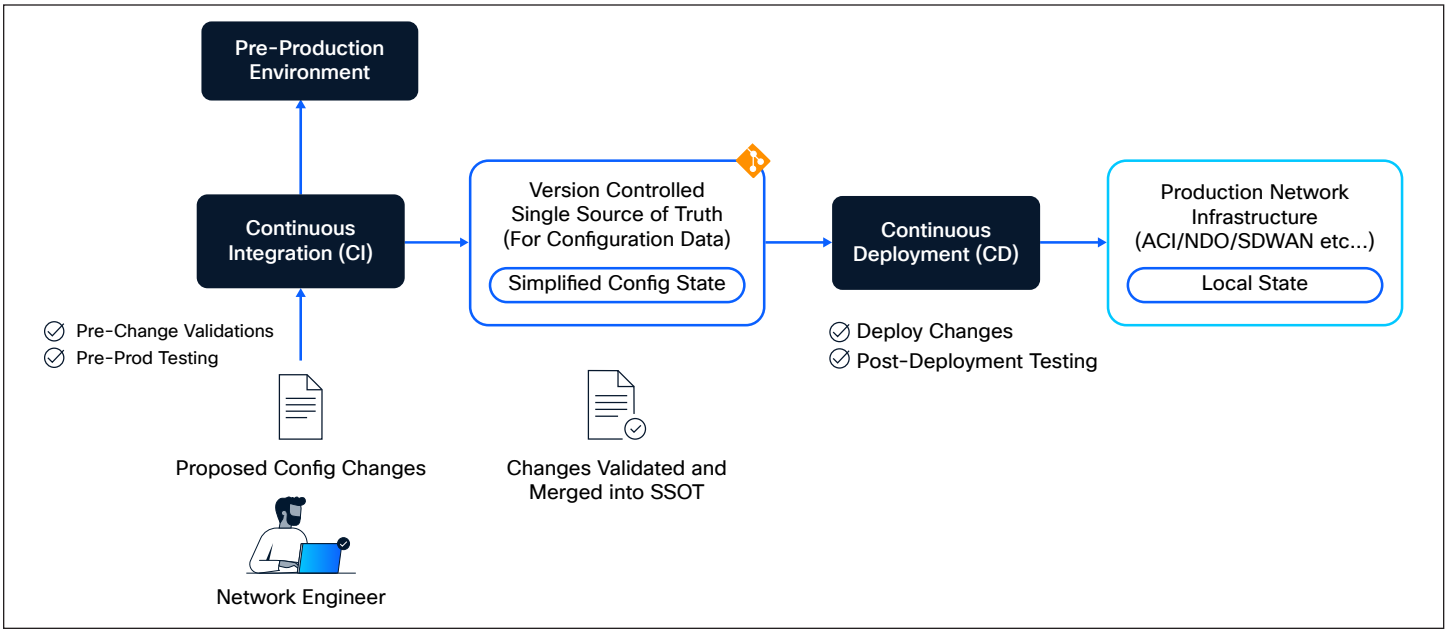


Figure 10. GitOps and CI/CD for Network Operations

© 2026 Cisco and/or its affiliates. All rights reserved.

This CI/CD workflow, when combined with GitOps practices, allows for a more streamlined and efficient approach to network operations. Here are some key benefits of implementing CI/CD in network automation:

- **Consistent Automation:** By combining single source of truth with CI/CD practices, all changes to the network configuration are made through a consistent and repeatable process. This ensures that every change is validated, tested, approved, and deployed using the exact same procedures, reducing the risk of human error and inconsistencies.
- **Compliance:** As all configuration changes in the network are performed through the single source of truth, the version control change log can be used to clearly document which changes were performed and by whom.
- **Easier Rollbacks:** In the event of a failure or issue with a deployment, having a version-controlled configuration coupled with CI/CD practices allows for easy rollback to a previous known good state. CI/CD practices ensure that a rollback goes through the same validation and testing processes as a regular deployment, ensuring the reliability of the automation.
- **Integrated Change Management:** The CI/CD process can integrate change management practices into network operations. Changes to the configuration (after it passes the CI stage) can be tracked, reviewed, and approved through pull requests, ensuring that all modifications are documented and auditable. This is particularly important for compliance and governance purposes.
- **Open and Extensible Framework:** Changes to infrastructure are often not limited to just one domain, or even just to networking. Adopting network automation using Cisco Network as Code and CI/CD practices allows for an open and extensible framework that can be extended to include additional dependencies, tests, or tasks (such as notifications). This leads to a more holistic approach to infrastructure management, where network changes can be coordinated with other components of the IT ecosystem, such as servers, applications, and security policies.
- **Automate Everything:** CI/CD practices encourage the automation of all aspects of the network operations process, from validation and testing to deployment and monitoring. This can be further extended to include automated documentation generation, compliance checks, and reporting. By automating these tasks, network teams can focus on more strategic initiatives, such as optimizing network performance or implementing new services.

Northbound Integration using NAC-API

Managing and configuring the network using Network as Code (NaC) are often not performed in isolation, and careful planning is essential to ensure seamless interaction between NaC and external systems. Northbound integration enables NaC, orchestration tools, and business applications to work together through APIs to manage the network configuration.

To facilitate and streamline northbound integration, Network as Code comes with its own API interface layer called **Network as Code API** (NAC-API), which is provided as a stateless container that provides two types of northbound APIs:

- **GraphQL APIs,** which primarily are used to query the network configuration stored in Git. The GraphQL API, for example, is leveraged by the Cisco AI Assistant for Services as Code.

© 2026 Cisco and/or its affiliates. All rights reserved.

- **REST APIs,** which primarily are used to perform add, modify, replace, and delete operations to the network configuration stored in Git.

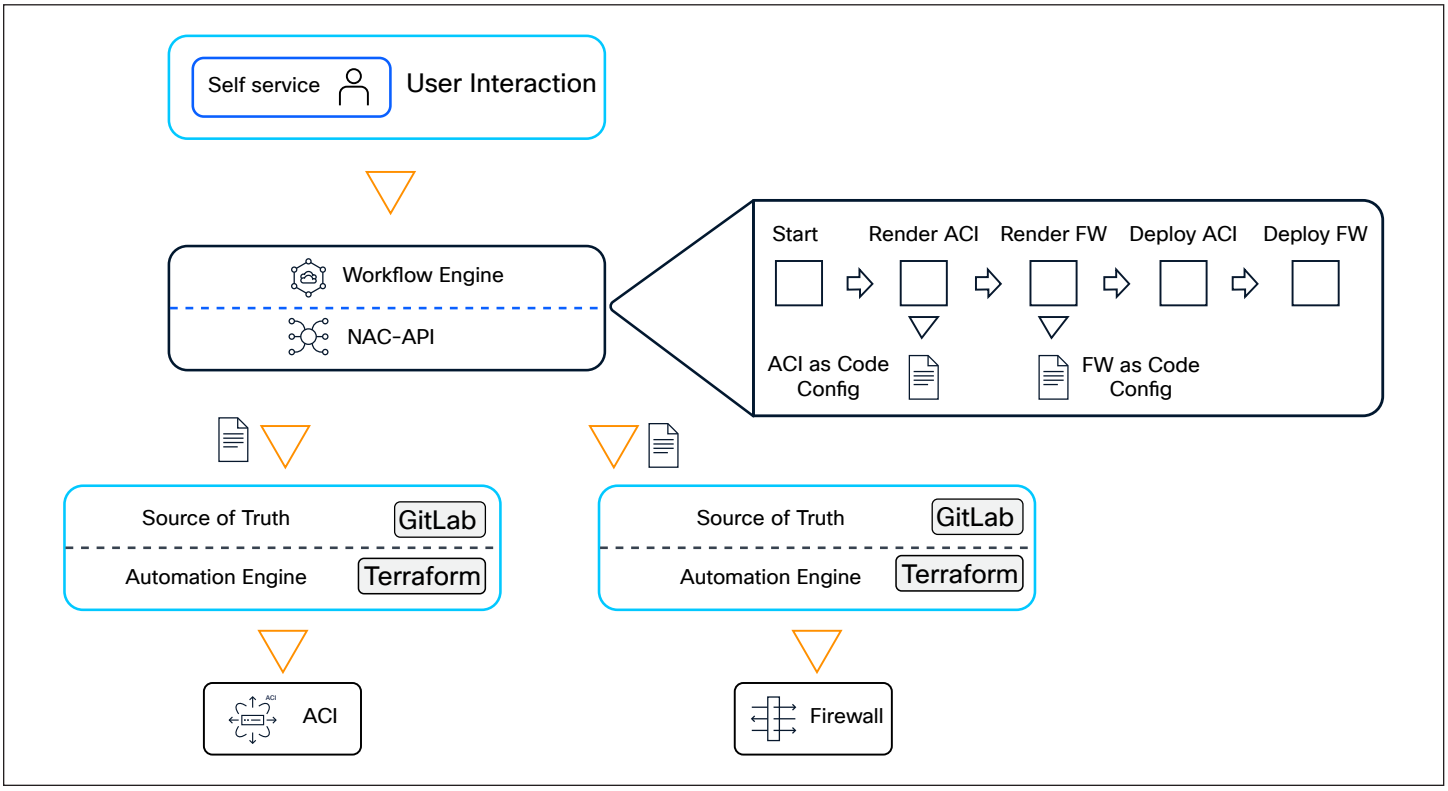


Figure 11. Northbound Integration using NAC-API

The NAC-API is designed to follow the same process as engineers would perform manually when performing changes to the intended configuration using Network as Code:

1. Create a new Git branch based on the main/master branch.
2. Make the required modifications to the data model (YAML files) and push the updated file(s) to the newly created remote branch.
3. Create a Pull Request / Merge Request for merging the changes into the main/master branch.
4. If/when desired, merge the Pull Request / Merge Request once the corresponding pipeline has finished successfully, thus resulting in the desired configuration changes getting applied to the network.

In terms of authentication and role-based access, the NAC-API currently supports bearer token authentication for both API types. Toward the Git repositories NAC-API also leverages bearer token authentication, only in this case the tokens are generated within the Git platform.

© 2026 Cisco and/or its affiliates. All rights reserved.

Deployment considerations

When designing and implementing a Cisco Network as Code (NaC) solution, it is important to keep several core principles since these will guide the transformation of network operations into a programmable, automated, and collaborative process:

- **Single Source of Truth:** As described, it is important to have a single source of truth when it comes to the network configuration. At least for part of the network configuration that is managed through NaC.

  This means that all configuration changes to the part of the network should go through NaC, effectively leaving the GUI and/or CLI of the network devices to read-only operations. Only in the rare occasion of severe network issues situations should the GUI/CLI of the network devices be used to restore connectivity, after which the NaC solution must be brought back into sync with the network configuration.
- **Separation of data and code:** NaC provides a clear separation of data, which represents the desired state of the network, from the code that configures the network. This distinction enables users of Network as Code solutions to utilize established data models and codebases, allowing them to concentrate on defining their unique configurations.
- **Test-Driven Automation:** With NaC, pre- and post-change validations are leveraged to identify and rectify potential configuration issues that may lead to outages as early as, and in most cases before, the change gets implemented in the network.
- **Open Architecture:** The NaC implementation is based on an open architecture that leverages open-source tools to ensure the maximum level of flexibility and extendibility. This open architecture enables NaC to integrate with existing tooling, whether it is existing Git and CI/CD solutions, existing secure vault solutions for storing sensitive data, or higher-level orchestrators such as Service Catalogues, etc.

These core principles are important to keep in mind when implementing the Network as Code implementation, since they allow the solution to grow and adapt over time—not only to additional network architectures, but also in terms of integration with other solutions.

Data Model Types and Deployment Scenarios

As described Cisco Network as Code provides different types of data models:

- Controller-Centric Data Models
- Solution-Centric Data Models
- Device-Centric Data Models

The choice of data model depends on the network infrastructure being automated. In cases where the network includes a tightly integrated controller, such as the APIC or Catalyst SD-WAN Manager, the appropriate controller-based data model should be leveraged. However, in scenarios where the controller is less tightly coupled with the network solution, the decision becomes less clear, prompting consideration of whether to adopt a solution-centric or device-centric data model.

© 2026 Cisco and/or its affiliates. All rights reserved.

Solution-Centric Data Model

To this point in this document, we have mainly presented the concepts of Network as Code in the context of Cisco controllers. These controllers play a direct role in the deployment of the configuration, and these controllers usually are focused in delivering a specific type of network configuration. If you were to look at Nexus Dashboard, it can deploy multiple types of network fabrics in the data center. These fabric types can include VXLAN/EVPN, External Border Gateway Protocol (eBGP), classic Ethernet, and more.

These controllers are performing a type of function that can be considered “intent”-based. Intent- based is different than device-based in various aspects, but the key element to understand is that intent-based is focused on the “what” instead of on the “how.” When utilizing Nexus Dashboard and requesting the configuration of VXLAN/EVPN, the controller performs actions to achieve the desired state without requesting the user to “define or declare” how the user would like to see the configuration.

Network as Code can perform this function directly in its intent-based approach for VXLAN/EVPN fabrics. How is this different? In a traditional device-based approach, you are focused on the specific commands that you need to run on the device to achieve a specific configuration. This is a very imperative approach, where you are telling the device exactly what to do. In an intent-based approach, you are focused on the desired state of the network, and automation configures this based on its defined best practices constructed by network engineers.

VXLAN/EVPN is designed around a spine/leaf (CLOS) fabric topology, which has commonality in the structure. In this methodology, network switches are given specific roles, including the types “spine”, “leaf,” and “border.”

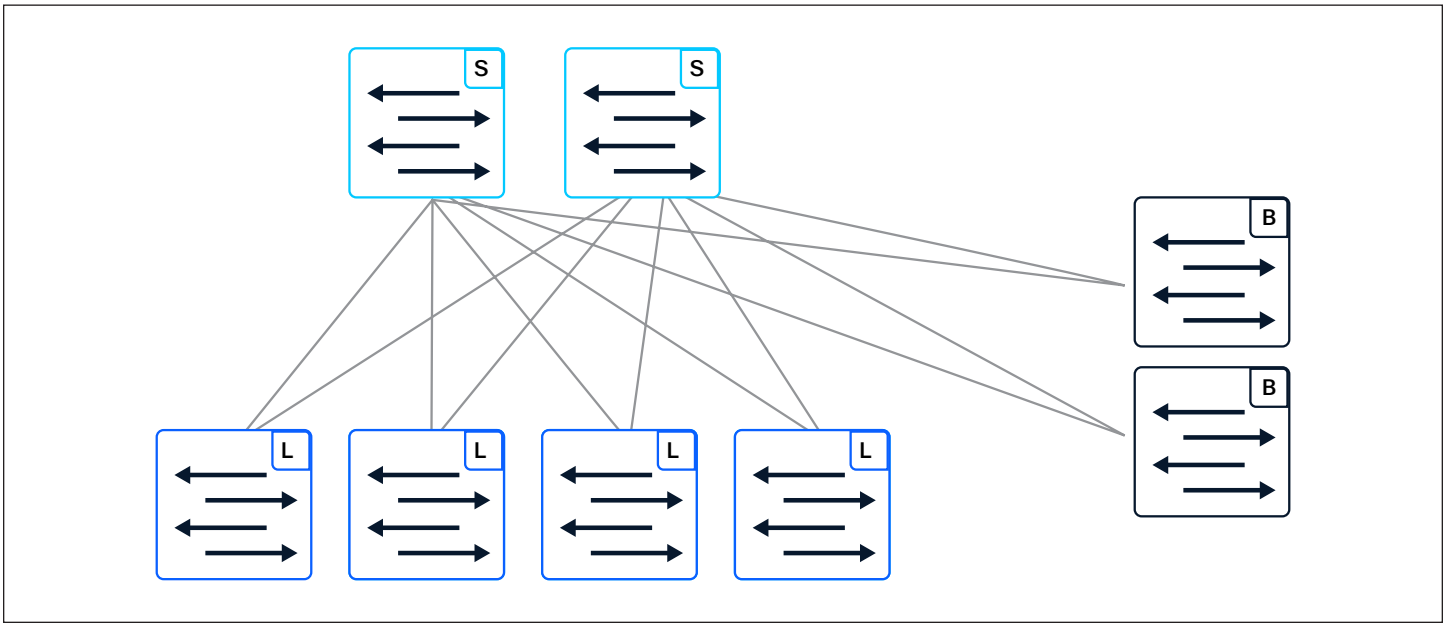


Figure 12. VXLAN/EVPN Fabric Topology

© 2026 Cisco and/or its affiliates. All rights reserved.

When applying configuration to this fabric type, there is a commonality in how the configuration is applied based on these roles. With the focus of two network layers (underlay and overlay), the configuration can be symmetric across the fabric. For automation, this commonality can facilitate implementation due to assumptions that can be made true on the configuration, especially on the underlay. For Network as Code, instead of focusing on each device configuration independently, we focus on the "intent" that is needed to configure the role for each device. A great example that is complex to do manually is the definition in large scale of route reflectors due to peering in the BGP configuration.

Route Reflectors

For the configuration of the VXLAN/EVPN fabric, we configure BGP to carry the endpoint routing information in the fabric. This configuration becomes a series of configuration of peering between the leaf nodes in the fabric and the spine nodes that will act as the "reflectors" of the requests from the leaf nodes toward the location of devices across the fabric. The abbreviated configuration would look like:

Spine Configuration (peers x leaf count)	Leaf Configuration (peers x spine count)

router bgp 65001
  router-id 10.1.1.1
  neighbor 10.1.1.11
  remote-as 65001
  description Peer to d2d-cml-leaf1
  update-source loopback0
  address-family l2vpn evpn
  send-community
  send-community extended
  route-reflector-client
	router bgp 65001
	  router-id 10.1.1.11
	  neighbor 10.1.1.1
	  remote-as 65001
	  description Peer to d2d-cml-spine1
	  update-source loopback0
	  address-family l2vpn evpn
	  send-community
	  send-community extended
	  route-reflector-client

Figure 13. IOS-XE BGP Configuration

In a typical "device" model approach, the operator designing the automation would be required to configure each leaf pointing to a defined spine that would be the route reflector. Each spine is also required to be configured to peer to each leaf node. In the intent-based model, the approach is different, as the intent is part of the automation construct.

© 2026 Cisco and/or its affiliates. All rights reserved.

When looking at the VXLAN/EVPN data model for Network as Code, we define each device with a specific role.

```
---
vxlan:
  topology:
    switches:
      - name: d2d-cml-spine1
        role: spine
      - name: d2d-cml-spine2
        role: spine
      - name: d2d-cml-leaf1
        role: leaf
      - name: d2d-cml-leaf2
        role: leaf
```

Figure 14. VXLAN as Code – Device/Role Definition

In this definition in the data model, we declare that one spine is the route reflector (spine1) in this network. That means that Network as Code will automatically build the relationship between the devices for this configuration. To accomplish this, Network as Code leaf nodes will create a BGP peering configuration that points to this single spine, and the spine will configure peering to all the defined leaf nodes part of the network.

In addition, if the network operator where to update the role of the second spine to:

```
---
vxlan:
  topology:
    switches:
      - name: d2d-cml-spine2
        role: spine
```

Figure 15. VXLAN as Code - Adding Spine Device

Upon execution of Network as Code, the automation will automatically add all the peering from the leafs to point to the second spine. In the device model approach the configuration would be on a per-device or device-group basis. This arrangement would then require updating and changing the configuration to these devices on a device- or group-basis based on the defined operator-declared configuration.

© 2026 Cisco and/or its affiliates. All rights reserved.

Underlay Routing

Another example of how “intent”-based models can alleviate complexity in configuration is the definition of the underlay routing protocol for a VXLAN/EVPN fabric. In this case the definition of the underlay routing is controlled in a single location in the data model.

```
---
vxlan:
  underlay:
    general:
      routing_protocol: ospf
    ospf:
      area_id: 0.0.0.0
      authentication_enable: false
      authentication_key_id: 127
      authentication_key: 9125d59c18a9b015
    isis:
      level: level-2
      network_point_to_point: true
      authentication_enable: false
      authentication_key_id: 127
      authentication_keychain_name: isis-keychain
      authentication_key: 071b245f5a58485744
      overload_bit: true
      overload_bit_elapsed_time: 60
```

Figure 16. VXLAN as Code - Underlay Network

Here in the data model, we define the routing protocol. Currently supported in the VXLAN/EVPN data model is either “ospf” or “is-is.” In addition, parameters required for each routing protocol are defined. Using intent-based model simply changing the one parameter definition of routing_protocol and running the automation would cause Network as Code to completely re-configure all the underlying fabric network routing protocol to the declared state.

Compare this to a device-centric model that would require the changes on each device to implement the configuration required. This would involve building the router process and then adding the process to every interface participant in the underlay in between the leafs and spines including the loopback interfaces.

Summary

When you compare the difference between a “device”-centric model and an “intent” model, you can observe the value of the intent-based model in accelerating the creation of defined network constructs that have “roles.” In the case of VXLAN/EVPN, the underlay in such a topology is very similar between the devices. This allows automation in Network as Code to simplify configuration of VXLAN/EVPN fabrics, focusing on simply defining the intent of the configuration and allowing the underlying model to perform the task.

Now that we can see the value of an intent-based approach, the device-centric approach still holds value in specific use cases, including models such as defining IPN network configuration for Multisite since these models are very specific to the device.

© 2026 Cisco and/or its affiliates. All rights reserved.

Device-Centric data model

The intent-based model offers great capability when the network requires a build that contains a focus on devices in the network having roles and symmetry. At times, another approach is needed, where the focus is simply pushing a configuration to a specific device. In the device-centric approach the focus moves from roles into groups.

In the previous example we used the example of a VXLAN/EVPN fabric. While the most optimal solution for this automation using Network as Code is to utilize an intent-based strategy, it is possible to accomplish a similar outcome using the device-centric approach with device groups. For some network operators, this method might be perceived as providing more “granular” control on how exactly they wish to configure the underlying fabric.

In this model, instead of focus on declaring that the network has an Open Shortest Path First (OSPF) underlay and allowing the automation to configure it, the operator builds the exact configuration that they would want on the devices. Next, the operator integrates a template approach into device groups to allow these to be applied across the network.

```
---
nxos:
  devices:
    - name: leaf1
      configuration:
        routing:
          ospf_processes:
            - name: UNDERLAY
              vrfs:
                - name: default
                  router_id: 10.1.1.101
                  areas:
                    - area: 0.0.0.0
    - name: leaf2
      configuration:
        routing:
          ospf_processes:
            - name: UNDERLAY
              vrfs:
                - name: default
                  router_id: 10.1.1.102
                  areas:
                    - area: 0.0.0.0
```

Figure 17. NX-OS as Code - Underlay Network

As you can see in this data model, in a device-centric approach the focus is on specifying how each device would be configured. There are two separate devices in this example: leaf1 and leaf2. These have the OSPF process enabled with the name UNDERLAY. As you can see, this scenario allows for the unique or bespoke definition of the OSPF process, which has the advantage of granularity and the disadvantage of potentially missing best practices that the intent model attempts to embed into the device configuration.

To mitigate extensive configuration construction, the NX-OS model for Network as Code provides two elements: the inclusion of device groups and templates.

© 2026 Cisco and/or its affiliates. All rights reserved.

Device groups

Using device groups, it would be possible to group similar devices and apply grouped configurations to them. When configuring OSPF for an NX-OS device, the same command for the underlay OSPF process is used across all the devices in the fabric. Utilizing groups, it would be possible to utilize groups to define a standard configuration that is then applied with defined per- device variables. In the following example we have the definition of a device named LEAF1 into the device group LEAFS.

```
---
nxos:
  devices:
    - name: LEAF1
      url: https://10.15.37.101
      device_groups:
        - LEAFS
      variables:
        hostname: LEAF1
        lo0_ip: 10.1.100.101
        vtep_ip: 10.1.200.101
    - name: LEAF2
      url: https://10.15.37.102
      device_groups:
        - LEAFS
      variables:
        hostname: LEAF2
        lo0_ip: 10.1.100.102
        vtep_ip: 10.1.200.202
```

Figure 18. NX-OS as Code – Assignment to Device Groups

The operator then assigns each device a unique set of values for generic interfaces. To simplify for this documentation, we can see that a variable definition for each switch is the configuration of the VXLAN Tunnel Endpoint (VTEP) IP address that is unique to each leaf in the fabric.

```
---
nxos:
  device_groups:
    - name: LEAFS
      configuration:
        interfaces:
          loopbacks:
            - id: 1
              interface_groups: [LOOPBACK_INTERFACE]
              ipv4_address: ${vtep_ip}/32
```

Figure 19. NX-OS as Code - Device Group Configuration

© 2026 Cisco and/or its affiliates. All rights reserved.

If you have any experience with VXLAN/EVPN, you might be asking where the definition of loopback0 is. The NX-OS model for Network as Code also is capable of creating a general configuration structure that is applied to all the devices in the network.

```
---
nxos:
  global:
    configuration:
      interfaces:
        loopbacks:
          - id: 0
            interface_groups: [LOOPBACK_INTERFACE]
            ipv4_address: ${lo0_ip}/32
```

Figure 20. NX-OS as Code – Global Configuration

As you have probably derived, every device in a VXLAN/EVPN fabric has the definition of loopback0 that is used for all peering connections for BGP. But the VTEP loopback is placed only on endpoints devices such as leafs and border gateways where tunnel traffic source and destination are established. For this reason, the loopback definition for VTEP is constructed in the LEAFs group to avoid the creation of the VTEP loopback interfaces in the SPINE devices.

In addition, for the NX-OS model for Network as Code, the interface_groups allow creating common configurations. In the case of loopback0, it will be part of a group defined for loopback interfaces that will then receive the configuration defined as:

```
    - name: LOOPBACK_INTERFACE
      configuration:
        ospf:
          process_name: UNDERLAY
          area: 0.0.0.0
```

Figure 21. NX-OS as Code – Interface Groups

Next the OSPF process is applied to those interfaces and assigned the standard UNDERLAY process name common for VXLAN/EVPN for functional OSPF configuration.

Summary

The device-centric approach provides the flexibility to configure network devices with specific configurations. When compared to an intent-based model such as the VXLAN/EVPN for Network as Code, it provides more flexibility. At the same time the burden is shifted to the operator on how to configure these different capabilities.

© 2026 Cisco and/or its affiliates. All rights reserved.

DevOps Practices and Branching Strategy

The adoption of Network as Code goes beyond automation, embracing modern DevOps principles to improve agility, consistency, and scalability in managing network configurations. By integrating practices such as collaboration, Continuous Integration/Continuous Delivery (CI/CD), and version control, network engineering teams can operate more effectively. At the heart of these workflows are Git and branching strategies, which enable streamlined version control and collaboration.

DevOps Practices in Network as Code

DevOps practices bring software development methodologies to the networking domain, enabling teams to treat network configurations as code. This approach allows for storing, versioning, and managing configurations in repositories, ensuring traceability, and reducing risks. Collaboration is enhanced through Git workflows, where teams can review and validate changes before deployment. Automated CI/CD pipelines play a key role by running pre-change validations, syntax checks, and simulations to catch errors early. Once validated, configurations can move seamlessly through the pipeline for deployment, minimizing manual intervention.

Git Branching Strategies for Network as Code

Branching strategies in Git are essential to manage parallel development, isolate changes, and ensure a stable production environment. Common strategies include:

- The **main branch** serves as the stable, production-ready version of the network configurations. All changes are tested and validated before being merged here.
- **Feature branches** allow engineers to work on specific updates or changes in isolation, reducing the risk of conflicts.
- **Hotfix branches** handle urgent production issues, providing a quick path to address critical bugs.

These strategies provide a structured approach to managing network changes, ensuring stability while allowing flexibility.

To ensure smooth workflows, branches should be kept short-lived to avoid merge conflicts and ensure relevance. Descriptive naming, such as `feature/vlan-restructuring` or `hotfix/bgp-routing-issue`, helps maintain clarity and organization. Code reviews are key for ensuring quality, while automated testing integrated into CI/CD pipelines can catch issues early. Regular merging of changes from the main branch into feature branches helps keep work aligned with the production state and avoids divergence.

© 2026 Cisco and/or its affiliates. All rights reserved.

Identifying and Selecting Toolchain

Identifying the tools for the Network as Code implementation and hosting them in the right place are important design decisions. As a customer, you have a variety of choices based on your comfort, expertise, and licensing requirements.

The components involved in are version control system including GitHub and GitLab; Continuous integration tools such as GitLab, Jenkins, working environment (runner) – VMs, Containers; and Collaboration tools—Webex, Slack, Microsoft Teams.

- **Customer provided (IT owned or Network dedicated):** In large organizations, version control and CI/CD tools are often already in place to support application development across the enterprise. As part of the IT infrastructure team, you can leverage this existing environment to run your Network as Code projects. Utilizing the organization's established tools ecosystem ensures alignment with enterprise-wide support, avoids the need to rebuild tooling, and helps mitigate additional licensing costs. This approach streamlines integration and maximizes the value of existing resources.
- **Cisco provided CI/CD Tools:** Alternatively, Cisco can support you in building the required tools to run Network as Code. Cisco offers a Common Automation Framework (CAF) virtual appliance which uses Docker for running Gitlab, Gitlab Runners, and other necessary tools that need to invoke the 'as Code' pipeline. CAF is Cisco Security Development Lifecycle (CSDL) certified and supports air-gapped networks also without need of internet connection.

Deployment and Security Considerations

To successfully deploy CI/CD pipelines for managing network configurations, two key factors to consider are network reachability and environment segmentation. These elements are critical for minimizing risks, enhancing confidence in automated deployments, and aligning network operations with modern DevOps delivery practices.

As Network as Code (NaC) becomes central to automating network changes, CI/CD pipelines play an essential role in streamlining these processes.

Network Reachability

Network reachability forms the backbone of successful automation. The CI/CD executor/runner must have secure and well-defined access to network devices (for device-centric data model-based automation) and controllers (for solution- or controller-centric data model-based automation). This connectivity is established using standard network management protocols, such as:

- Secure Shell (SSH)
- NETCONF
- RESTCONF
- REST APIs
- gRPC Network. Management Interface (gNMI)

© 2026 Cisco and/or its affiliates. All rights reserved.

Management access should be isolated from production traffic for security and stability. This isolation can be achieved using techniques such as:

- Virtual Routing and Forwarding (VRFs)
- Access Control Lists (ACLs)
- Jump Hosts

Furthermore, firewall rules must explicitly allow necessary control-plane communication. Wherever possible, Out-of-Band Management (OOBM) should be utilized to enhance reliability during testing or in the event of network failures, by providing an independent management path.

Environment Segmentation

Environment segmentation is essential for safely testing, especially major or non-standard network changes prior to their deployment in a production environment. It enables the validation of configurations and the simulation of their impact, ensuring that existing services remain unaffected during the testing process.

Effective environment segmentation may include:

- **Digital Twin or Staging Lab:** Utilizing a digital twin or a dedicated staging lab that accurately mirrors the production topology provides a safe sandbox for validation in the case of major changes.
- **Automated Checks:** Integrating automated checks into the CI/CD pipeline is vital for verifying configurations and simulating potential impacts.
- **Git Repositories or Branching for Environment Isolation:** Using separate Git repository for the different environments or using Git branches is a common practice, with examples including dev (development), staging (pre-production testing), and main (production).
- **Secure Secrets Management:** To maintain security and integrity, secure secrets management should be used to isolate credentials and inventories per environment, preventing cross-environment credential leakage.

By meticulously designing CI/CD pipelines with robust network reachability and clear environment segmentation, organizations can significantly enhance their capabilities, reduce deployment risk, and increase confidence in automated network changes.

Security considerations

Within infrastructure automation, network automation is often considered the most sensitive, complex, and challenging domain. This sensitivity arises from the fact that network configurations directly impact the availability, performance, and security of the entire IT infrastructure. As such, it is vital to ensure that network automation practices are secure, reliable, and resilient.

Below are some key security considerations for network automation, and how Network as Code practices can help address these considerations.

© 2026 Cisco and/or its affiliates. All rights reserved.

Credential management

Network devices often require sensitive credentials for access and configuration. It is essential to manage these credentials securely, ensuring that they are not hard-coded in scripts or stored in insecure locations. This will not only protect against unauthorized access but also ensures that credentials are rotated regularly to minimize the risk of compromise.

The CI/CD pipeline can be integrated with secure vaults (e.g., HashiCorp Vault [14], GitLab/GitHub variables, etc.) to manage sensitive credentials. This ensures that credentials are not hard-coded in the automation scripts and are accessed securely during runtime. This is achieved by fetching the credentials from the vault at runtime, ensuring that they are not stored in the code repository, even in an encrypted form.

The credentials stored outside of the code repositories can be injected into Network as Code solutions two ways:

- Expose the credential as an environment variable at runtime, which allows placeholder values in the data model to reference the name of this environment variable.
- If the vendor of the secure vault provides a Terraform provider for the vault solution with data resources to retrieve the credentials, then this approach can be used for the configuration management part. Retrieving credentials through a Terraform provider does however not work for pre-change validation and post-deployment testing where the environment variable path still needs to be pursued.

Access Control Management

Any network automation platform must be securely configured to ensure that only authorized personnel have access to the automation tools and the underlying network devices. This includes implementing Role-Based Access Control (RBAC) and ensuring that access is granted based on the principle of least privilege.

Platforms like GitLab and GitHub provide robust access control mechanisms, allowing teams to define who can access the automation repositories and what actions they can perform. This includes implementing RBAC, where users are granted permissions based on their roles, ensuring that only authorized personnel can make changes to the network configuration.

Protected Execution Environment

Automation tools must have access to the network devices only while running automation tasks. Once the task is completed, the access should be revoked. This temporary access model helps in minimizing the attack surface and reducing the risk of unauthorized changes to the network configuration.

For Network as Code, it is generally recommended to use container-based execution environments for running automation tasks. Containers are created with the necessary tools and dependencies, and they will retrieve the required credentials required to access the network infrastructure at runtime. Once the task is completed, the container is destroyed, ensuring that access to the network devices is temporary and controlled.

© 2026 Cisco and/or its affiliates. All rights reserved.

Protect Potions of Configuration

Often it is important to protect certain portions of the network configuration from being modified. This will ensure that critical configurations, such as security policies or routing protocols, are not inadvertently changed.

This can be achieved in different ways. For example, using GitOps practices, certain portions of the network configuration can be protected by defining policies that restrict changes to specific files or directories in the repository. This ensures that critical configurations are not modified without proper review and approval. Platforms like GitLab, GitHub etc., provide features such as Branch Protection (for controlling changes that merge into main branch), Code Owners (a GitLab feature), which allows teams to define who can approve changes to specific files or directories, adding an additional layer of protection.

Obfuscation of Sensitive Data

Sensitive data, such as passwords or API keys, should be obfuscated or encrypted in the automation execution environment. This will help protect against accidental exposure of sensitive information during the automation process.

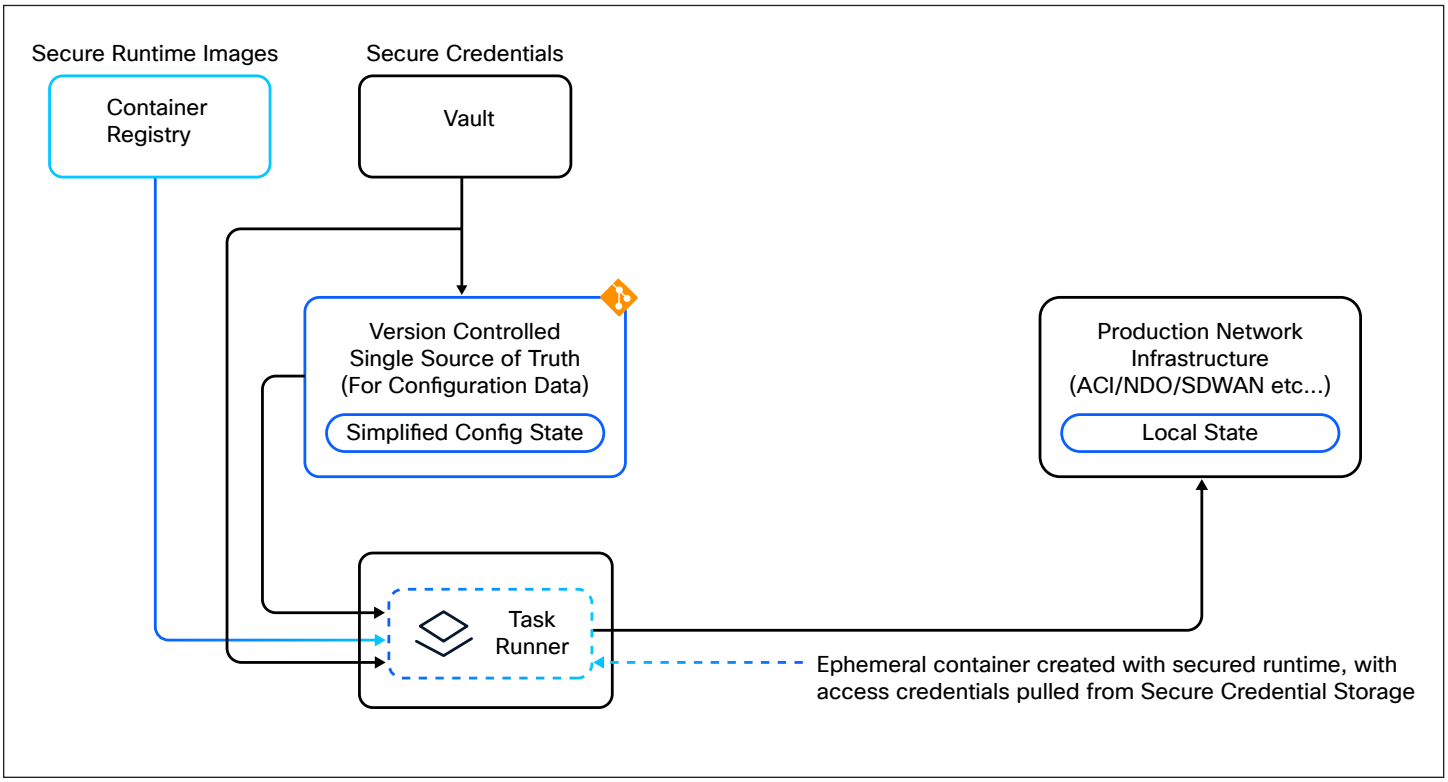


Figure 22. Secure CI/CD Execution Environment

In the CI/CD pipeline, sensitive data can be used in the automation tasks securely by using environment variables or secret management features provided by the CI/CD platform. This ensures that sensitive information is not exposed in logs or test results and is only accessible to the automation tasks that require it.

© 2026 Cisco and/or its affiliates. All rights reserved.

Greenfield vs Brownfield approach

Adopting a structured and well-thought-out data model is a foundational and recommended approach when designing a new network architecture. This methodology ensures alignment with industry best practices and guidelines, such as those outlined in Cisco Validated Designs (CVDs) and Cisco Validated Profiles (CVPs). A new implementation or migration to a modern architecture presents an opportunity to incorporate new or updated design principles, ensuring key aspects are thoughtfully considered from the outset.

The Data Model–driven approach offers a robust framework that translates network design into an easily interpretable and actionable implementation plan. This approach facilitates the identification of inconsistencies or deviations from the intended network architecture. For migration scenarios, it enables a straightforward comparison between pre- and post-migration states, enhancing visibility and reducing complexity during transitions.

Cisco's Network as Code framework provides comprehensive examples of Data Models tailored for various architectures, whether they are solution-centric, controller-centric, or device-centric. These reference models simplify the process of building greenfield environments by offering validated, supported templates that accelerate deployment while adhering to best practices.

Challenges in Brownfield Networks

While greenfield deployments often benefit from clean-slate planning, most organizations face the challenge of working with organically evolved networks built over years. These environments often include inconsistent configurations and designs that are difficult to reverse-engineer into a cohesive data model (example: duplicated object names, incomplete configurations, etc.). Representing such networks accurately requires significant effort, and there is a risk of losing critical details during the translation process. This complexity, along with associated costs and risks, often makes organizations hesitant to adopt the Network as Code framework.

Transitioning to Network as Code in Brownfield Scenarios

In brownfield deployments, transitioning to a Network as Code operational model necessitates addressing two critical steps:

1. **Exporting the Current Configuration and Translating it into a Data Model**

   If the existing infrastructure supports APIs, the process involves extracting the current configuration through a series of API calls and organizing it into a comprehensive data structure that represents the production environment. A controller-based approach can significantly streamline this process, as it allows a single point of API interaction, avoiding the complexity of retrieving configurations from hundreds or thousands of devices.

   Once the configuration is gathered, it must be translated into one or more Data Models, depending on the deployed solutions and environments. While automation can handle much of this process, manual validation and adjustments may still be required. To facilitate these tasks, Cisco's Network as Code framework provides tools such as nac-collector and nac-tool, which assist in collecting and structuring the required data.

© 2026 Cisco and/or its affiliates. All rights reserved.

2. **Importing the Data Model into the Framework's State**

   For Day-N network operations, the framework must track the current state of the network to ensure that only incremental (delta) changes are provisioned. In large-scale environments, operating without knowledge of the existing configuration is not practical, and, in some cases, infeasible. Importing the Data Model into the framework ensures that all relevant information about the network's current configuration is captured. Once the organization transitions fully to the Network as Code framework, these processes can be executed seamlessly.

Phased Migration Strategy

To mitigate risks and ensure operational continuity, it is advisable to adopt a phased approach when migrating to the Network as Code framework. For example, organizations can start with the Data Center (DC) environment, followed by the Wide Area Network (WAN), and then campus networks. Within each environment, it may also be beneficial to manage specific controllers or individual sites selectively during initial adoption phases.

A key principle of the transition is ensuring that, once fully migrated to the Network as Code framework, no manual changes are made to the network. This ensures consistency, predictability, and alignment with the automated operational model, ensuring that the data model is the single source of truth.

Hybrid Operations: Balancing Automation and Manual Changes

For organizations adopting a Network as Code framework, one of the most critical principles is establishing a single source of truth and consistently applying it to all subsequent network changes. Once the framework is fully implemented, all modifications to the network are tracked and aligned with internal change management processes, ensuring operational consistency and auditability.

However, challenges can arise when out-of-band changes are introduced—those not reflected in the centralized data model. These changes might occur due to operators who have not fully adopted the new operational framework or during critical situations requiring emergency changes implemented through alternative channels, such as the console or a graphical user interface.

When out-of-band changes are made, the next execution of an implementation tool will identify discrepancies (delta) between the data model and the actual network configuration. Since the data model serves as the authoritative source of truth, the tool will attempt to overwrite these changes to restore alignment. This behavior underscores the importance of keeping the data model up to date and avoiding any unauthorized or unmanaged changes outside the framework. Operating in a hybrid approach, where both manual and automated processes coexist, can lead to significant network inconsistencies and disruptions in operational workflows.

© 2026 Cisco and/or its affiliates. All rights reserved.

Handling Emergency Changes: Exception Management

Although the ideal state with Network as Code is to operate exclusively through its methodologies, there may be scenarios where critical emergency changes must be applied manually.

Once stability is re-established, it is imperative to reconcile the manual changes with the Data Model to maintain its accuracy. This can be achieved either through manual updates or by leveraging Cisco CX's brownfield capabilities that allow for extracting the current configuration and synchronizing it with the Data Model. However, this process often requires careful validation and manual intervention. Given the complexity and potential for errors, emergency manual changes should be treated as exceptions, not as a recurring operational practice or a hybrid operating model.

Organizations should prioritize fully transitioning to the Network as Code framework and aim to eliminate reliance on manual, out-of-band changes. Maintaining the integrity of the data model as the single source of truth is critical to achieving consistent, reliable, and automated operations. Emergency changes should be carefully managed as exceptions, with processes in place to promptly update the Data Model and minimize the risk of operational inconsistencies.

Scaling Network as Code Implementations

When designing and implementing Network as Code, is it important to consider the size of the configuration—not only at the time of initial implementation but also the expected growth in the foreseeable future, for example at least the next 6, 12, 24, or 48 months.

The size of the configuration is important, as several challenges may arise as you add more resources to your configuration:

- The Terraform state file becomes bigger, and making changes with Terraform takes longer.
- A single shared state file is a risk. Making a change in a Development tenant could have implications to a Production tenant.
- No ability to run changes in parallel. Only one concurrent plan may run at any given time as the state file is locked during the operation.

These problems are not unique to Network as Code and would occur when scaling any Terraform resource using a single state file. To address these problems, the automation design must consider distributing state.

Understanding State Distribution

There is no golden rule as it relates to Terraform state distribution, and there are several considerations to be made around the amount and type of resources. Such analysis will help to determine how to best carve the state into multiple state files depending on the configuration and scale requirements.

© 2026 Cisco and/or its affiliates. All rights reserved.

While there is no limitation to the size of the configuration when it comes to Network as Code or Terraform itself, experience shows that execution times tend to increase significantly once the state file exceeds roughly 5000 entries (for ACI deployments). The generic recommendation is therefore to stay below this number and instead distribute the combined state over multiple state files.

Although the network state is divided into multiple state files with Network as Code, this does not necessarily mean that the desired configuration is equally divided. In fact, in most cases it is preferable to keep the desired configuration (YAML files) unchanged, and simply split the configuration into multiple Terraform state files using control knobs built into the Network as Code Terraform Modules.

```
$ tree -L2
├── data
│   ├── tenant_TEN1.nac.yaml
│   └── tenant_TEN2.nac.yaml
└── workspaces
    ├── tenant_TEN1
    └── tenant_TEN2
```

Figure 23. Network as Code – File Structure

The figure above illustrates one way of structuring the relevant files when dividing the state into multiple state files. It shows the desired configuration left untouched in the data directory, which means that there is no change for the engineer using Network as Code to manage in the network, regardless of whether a single or multiple state files are used.

```
module "aci" {
  source  = "netascode/nac-aci/aci"
  version = "1.0.1"

  yaml_directories = ["../../data"]
  manage_tenants   = true
  managed_tenants = [“TEN1”]

  write_default_values_file = "defaults.yaml"
}
```

Figure 24. Network as Code – Splitting Terraform State per ACI Tenant

The logic that splits the state into multiple files are in this example located within the main.tf files. In the case of ACI as Code, the Network as Code Terraform module provides several control knobs that can be used to manage each part of the ACI configuration that a given Terraform workspace will manage. In the above example is the module instructed to manage the ACI tenants specified in the “managed_tenants” list. In other words, with the specified configuration are the objects related to ACI tenant TEN1 store in its own state file, and similar configurations would be used for storing the other tenants and fabric configurations.

© 2026 Cisco and/or its affiliates. All rights reserved.

Another example would be to split states between logical modules such as a separate workspace for access policies, and a separate workspace for interface configuration. This type of setup can be also achieved using control knobs in the module configuration.

```
module "aci" {
  source  = "netascode/nac-aci/aci"
  version = "1.0.1"

  yaml_directories = ["../../data"]
  manage_access_policies   = true

  write_default_values_file = "defaults.yaml"
}
```

Figure 25. Scaling Network as Code - Transform State for ACI Access Policies

```
module "aci" {
  source  = "netascode/nac-aci/aci"
  version = "1.0.1"

  yaml_directories = ["../../data"]
  manage_interface_policies   = true

  write_default_values_file = "defaults.yaml"
}
```

Figure 26. Network as Code - Terraform State for ACI Interface Policies

Splitting configuration into multiple state files also implies that the CI/CD platform must support such setup. If the number of state files and thereby Terraform configurations (often referred to as workspaces [15]) iincreases significantly, it would not be efficient to run the jobs sequentially. With multiple workspaces, each one requires its own initialization, plan, and apply, which means each one needs to have a dedicated job or part of a job executing it. In most cases, it is possible to run such jobs in parallel. For example, tenant TEN1 configuration usually doesn't depend on TEN2 configuration, so if both tenants require a change, they could run in parallel. Correspondingly, if only tenant TEN1 is configured, there is no need to run jobs for tenant TEN2, which means jobs should be configured in a way that they would only be triggered when needed, not always.

CI/CD platforms typically have runners or executors, which execute jobs and scripts, and a single runner handles a single job. To run multiple jobs in parallel, it is necessary to plan the platform capacity adequately. Having a single executor means that all jobs would have to run sequentially anyway, and having too many executors could mean that most of them are unused resources. Depending on the CI/CD platform of choice (i.e. Gitlab, Jenkins, etc.) the number of runners or executors either can be statically defined, or they support the concept of runners fleet, where runners can get dynamically scaled according to the demand.

With the growing number of resources, it becomes even more important to select efficient Terraform backends to hold the state files. By default, the Terraform state is written in a local file, which has a very limited performance and is generally not recommended for use in production deployments. Remote Terraform backends such as AWS S3 bucket, Azure Blob Storage, Postgres database, or HTTP-type backends support more capabilities for

© 2026 Cisco and/or its affiliates. All rights reserved.

securing our state file [16]. Keeping the state file in a centralized remote location allows teams to collaborate on the same deployment, track changes, and add ability for state locking to prevent conflicts between concurrent runs. Although they require additional configuration and may add some overhead due to network communication, remote backends have typically much better performance in read/write operations than local files do. In a deployment with several thousand resources in a workspace, a single add or change operation might take hour(s) to plan and execute, while with remote backend this time is reduced to minutes.

More information about how to scale Network as Code deployments can be found in the scaling section of the Network as Code website [17].

Customer case studies

This section highlights real-world examples of organizations successfully implementing Network as Code to transform their network operations. Through these case studies, we explore the challenges they faced, the solutions deployed, and the measurable outcomes achieved. These stories provide practical insights and demonstrate the tangible benefits of adopting Network as Code in diverse environments.

Financial Services Customer – Large-Scale Enterprise Data Center

The customer operates within the financial vertical, with a global presence across the Americas, EMEA, and APAC regions. Their network environment is large and spans multiple domains; however this case study focuses on the Data Center infrastructure. A primary objective for the organization is to ensure uninterrupted network operations, which is critical to providing financial services to their customers.

Given the size and complexity of their Data Center network, one of the operational challenges for the customer is to maintain a sufficiently high velocity and success rate of changes to match the needs from their business entities. Network changes have historically been performed manually, which has resulted in lack of consistency in network configurations. This in turn increases the network complexity and time to resolution in case of issues or outages.

Challenges Overview

The following challenges highlight the operational and scalability concern faced by the customer:

- Large-scale environment with a high number of changes due to the environment constantly expanding and changing. The changes include provisioning of new leafs, switchports, and networks/subnets.
- The customer's operation is highly sensitive to Data Center disruptions, making uninterrupted operations essential.
- Managing the network configurations manually leads to inconsistent configurations due to human errors, which in turn have recently led to outages.
- Single change requests can include hundreds of new networks or interface configurations, which are inefficient and error-prone to execute manually.
- The absence of robust processes to validate configurations and perform testing during change execution increases the risk of misconfigurations and outages.

© 2026 Cisco and/or its affiliates. All rights reserved.

How Network as Code provides value

With the introduction of Network as Code the customer has been able to automate configuration related changes in their Data Center fabrics, integrating a robust approval process and ensuring consistency and validation across all changes.

During the change process, every configuration is rigorously tested for both semantic and syntactic correctness using the Network as Code framework, effectively eliminating the risk of introducing potentially disruptive changes. Once a change is deployed, the framework performs post-deployment testing to validate the operational health and stability of the network, ensuring that the network continues to perform as expected.

Deployment Architecture

The deployment architecture for this customer is built leveraging Network as Code for ACI and Cisco Nexus Dashboard Orchestrator (NDO). Multiple ACI fabrics and multiple NDO instances are separated using their own repositories, and inside the repositories each solution is further logically separated into folders that correspond to workspaces. Such separation was done having in mind the massive scale of configurations in this environment.

Multiple workspaces not only provide faster execution times, due to lowered number of resources in a state file, but also allow for isolating different parts of configuration such as tenants making sure that change in one will not affect another.

The structure below illustrates an example setup with two ACI fabrics and one NDO managing both fabrics:

```
$ tree -L2
├── ACI-x/
│   ├── data/
│   └── workspaces/
│       ├── tenant-A
│       ├── tenant-B
│       ├── access-policies
│       ├── interfaces-pod1
│       └── interfaces-pod2
│
└── NDO/
    ├── data/
    └── workspaces/
        ├── schema-A
        └── schema-B
```

Figure 27. Repository Structure for Large Scale Data Center Deployment

The data folder holds all the YAML configuration files and workspaces logically isolates configuration to different Terraform state files. NDO being the orchestrator of ACI fabrics will have majority of configuration of tenants inside schemas. Once schemas are deployed, this configuration is automatically pushed to ACI fabrics. It is therefore not necessary to manage the same configuration on ACI using YAML files, exception being certain objects that require site-specific configurations that are not supported by NDO.

© 2026 Cisco and/or its affiliates. All rights reserved.

Changes in the data model files are automatically detected by the CI/CD system and trigger respective jobs in a pipeline to deploy configuration only for the affected workspace(s). A typical change does not affect more than 5 workspaces at the same time, so the number of CI/CD pipeline runners is limited to no more than 10 per repository. Each change is first validated syntactically and semantically to verify that there are no typos or missing references, then Terraform runs the speculative plan, which is reviewed before the maintenance window to ensure that only expected changes are happening.

The CI/CD pipeline system is integrated with the service catalog, where engineers select services with templated configuration, ensuring that all the new configurations adhere to the standards.

During a maintenance window, the change is accepted and merged, triggering deployment to the APIC and/or NDO, after which the testing phase runs post-change validation. The entire change process is safeguarded at multiple stages, in an automatic manner, to rule out common issues like human errors. With this approach, execution times are significantly lower and include full validation of changes.

Value Proposition and Outcome Achieved

The automation architecture based on Network as Code has been transformative to the approach to managing the large-scale data center network.

By implementing this architecture, the customer achieves the following tangible outcomes:

- **Improved Efficiency and Accuracy** by automating all configuration changes in the network, eliminating the need for manual changes.
- **Improved Change Success Rate** by leveraging configuration validation and automated testing as a key component of the automation solution to ensure successful changes.
- **Improved Configuration Consistency and Compliance** across multiple products through pre-deployment validation enforcing best practices and proven configuration patterns.
- **Improved End-to-End Time to Implement** and test change during a maintenance window, which results in shorted maintenance windows and less service disruptions.
- **Shortened Time for Onboarding** new members of operational team by keeping all configuration centralized, in a human readable format and templated service catalog.
- **Leveraged Cisco AI Assistant t**o help understand and develop new network configurations outside of the service catalog.
- **Fully documented history of changes** by using embedded capabilities offered by using single source of truth.

© 2026 Cisco and/or its affiliates. All rights reserved.

Global Manufacturing Vertical – IT/OT Cross-Domain Automation

The customer operates within the manufacturing vertical, with a global presence across the Americas, EMEA, and APAC regions. Their network environment is large and spans multiple domains, including Campus, Wide Area Network (WAN), and Data Center infrastructures. A primary objective for the organization is to ensure uninterrupted network operations for both Information Technology (IT) and Operational Technology (OT), which are critical to sustaining production.

The customer's annual growth rate is projected at 10–15%, driving the need for scalable and resilient network operations. To address the critical nature of their environment, the customer has opted to segment their network operations by establishing distinct domains in each region. While this approach improves fault isolation and regional independence, it introduces significant operational overhead, as each region is managed by separate network controllers for Campus, WAN, and Data Center environments. In some cases, due to scale constraints, multiple controllers are required to manage a single regional domain.

Additionally, each region maintains a separate set of security components, including Network Access Control (NAC) systems and multiple Network Enforcement Points. This fragmented operational model has become increasingly challenging as the customer continues to expand, provisioning new production sites and deploying additional applications. The operational team is under growing pressure to ensure the network remains secure, consistent, and reliable, despite the added complexity. Furthermore, the need to test and replicate every change across three separate domains has led to significant resource investments to expand the network operations team.

One of the most critical operational challenges for the customer is the streamlined deployment of network changes across multiple regions and domain controllers. For example, implementing a new network segment or project to support access to a newly deployed application requires coordination across all domains. Specifically, the change involves:

- **Campus Network:** Provisioning a new network segment to accommodate the application.
- **WAN:** Extending the newly provisioned network across the WAN to ensure connectivity.
- **Data Center:** Configuring the new segment in the Data Center (e.g., creating a Bridge Domain).
- **Security Components:** Updating all relevant security policies, including Network Access Control (NAC) configurations and firewall rules, across both the Campus and Data Center domains.

This end-to-end process is complex, requiring close coordination between teams and tools across regions and domains. Without a high-level orchestration mechanism, the risk of configuration inconsistencies, delays, or security gaps increases significantly. The fragmented manual approach exacerbates these challenges, making it difficult to achieve standardization and operational efficiency at scale.

© 2026 Cisco and/or its affiliates. All rights reserved.

Challenges Overview

The following challenges highlight the operational and scalability concerns faced by the customer:

- **Large-Scale Environment with High Growth Projections:** The customer's environment is expanding rapidly, with annual growth expected at 10–15%. This includes the provisioning of new production sites and the deployment of additional applications, putting strain on existing resources.
- **Critical Nature of the Environment:** The customer's manufacturing operations are highly sensitive to any network disruptions, making uninterrupted operations essential.
- **Restricted Maintenance Windows:** Due to the highly critical nature of the environment, the number of planned maintenance windows is severely limited. As a result, every network change must undergo thorough testing and validation before deployment to ensure minimal disruption to production operations.
- **Regional Segmentation:** The network is split into distinct regional domains, each managed independently. This leads to challenges in replicating configurations consistently across regions.
- **Multiple Controllers per Region:** Due to the scale of operations, some regions require multiple controllers to manage the network, increasing complexity.
- **Highly Secure Environment:** The environment includes many security enforcement points and NAC components, creating operational challenges in maintaining consistency and reliability.
- **Inconsistent Configuration:** Recent network outages have been attributed to inconsistencies in configuration across domains.
- **Lack of Standardization:** The absence of standardized processes and configurations across regions has exacerbated operational inefficiencies and increased the risk of errors.
- **Insufficient Configuration and Security Policy Compliance Validation:** The absence of robust mechanisms to validate configurations and enforce security policy compliance increases the risk of misconfigurations and potential vulnerabilities. This lack of validation undermines the consistency, reliability, and security of the network.

© 2026 Cisco and/or its affiliates. All rights reserved.

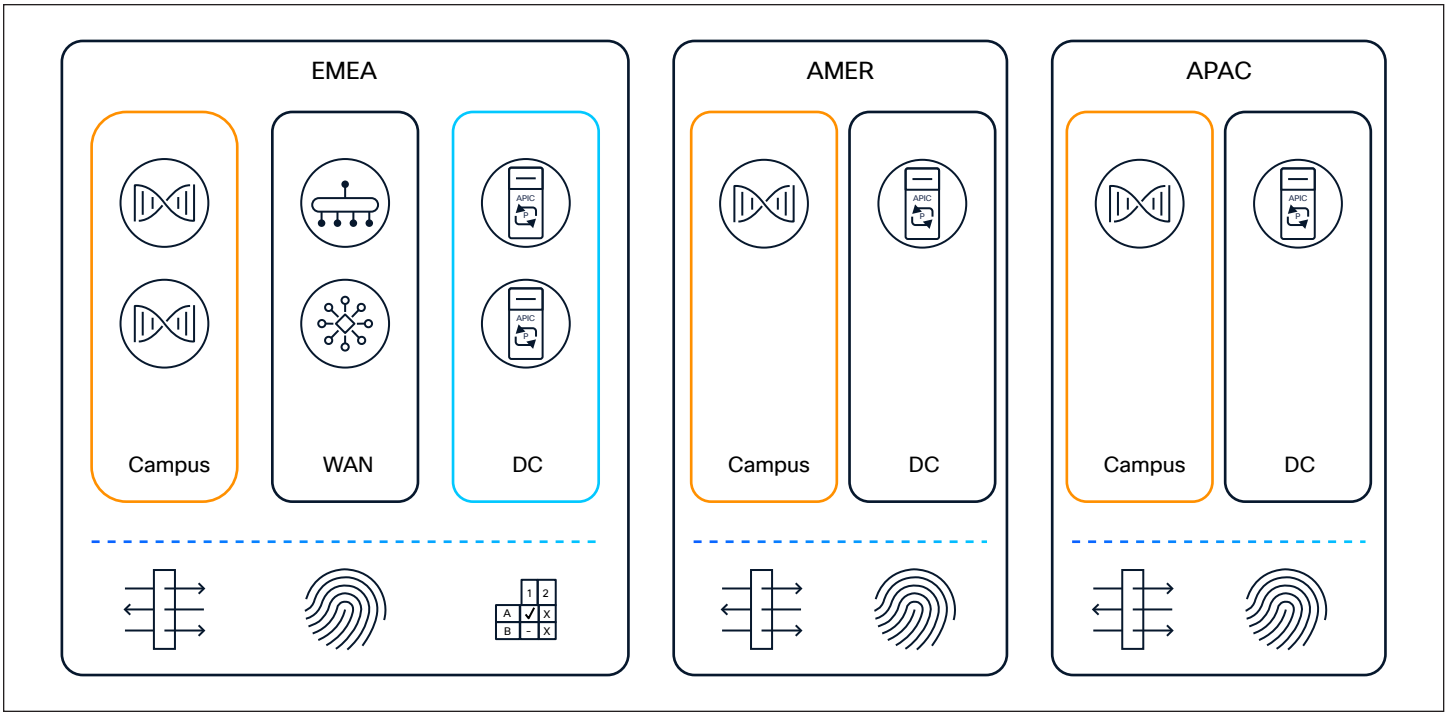


Figure 28. Multi-Architecture Deployment using Network as Code

How Network as Code provides value

The primary principle behind adopting the Network as Code methodology is to streamline network operations by automating most tasks, integrating a robust approval process, and ensuring consistency across all changes. This approach leverages a Data Model-driven framework, enabling organizations to replicate network changes in a controlled test environment, fully validate them through all phases of implementation, and subsequently apply them to the production environment. The Data Model-centric approach guarantees that changes implemented in production are identical to those tested, ensuring consistency and minimizing the risk of errors.

During the change process, every configuration is rigorously tested for both semantic and syntactic correctness, effectively eliminating the risk of introducing potentially disruptive changes. Once a change is deployed, the framework performs post-deployment testing to validate the operational health and stability of the network, ensuring that the network continues to perform as expected.

An additional critical advantage of the Network as Code methodology is its ability to maintain a comprehensive record of changes. All changes performed through the framework are tracked and auditable, enabling organizations to align with compliance requirements, improve traceability, and enhance operational visibility. By automating the entire lifecycle of network changes—testing, deployment, and validation—the methodology not only simplifies operations but also enhances the reliability and security of the network infrastructure.

© 2026 Cisco and/or its affiliates. All rights reserved.

The Network as Code methodology also effectively addresses the challenges of managing multi-controller environments, ensuring consistent configurations across all controllers. With Network as Code, any network change can be easily replicated across multiple controllers, maintaining full alignment and eliminating discrepancies between regions or domains.

For example, consider a change to a security component, such as adding a new firewall rule. Using the Network as Code framework, this rule can be seamlessly replicated across all regions, ensuring a consistent and robust security policy. By automating and standardizing such changes, the framework significantly reduces the risk of gaps or inconsistencies in security configurations. This capability is particularly critical for organizations in the manufacturing vertical, where operational security is paramount to safeguarding production environments and preventing disruptions.

This streamlined approach to multi-controller management not only strengthens the overall security posture but also reduces the operational overhead associated with manual replication of changes, allowing the network operations team to focus on higher-value tasks.

The introduction of northbound integration within the Network as Code framework enables seamless orchestration of changes across multiple domains. By leveraging a single northbound interface, organizations can implement a high-level orchestration layer or workflow engine where services are pre-defined and pre-configured. This approach streamlines the execution of complex, cross-domain changes, reducing operational complexity and improving efficiency.

With northbound integration, low-level CLI configurations are no longer required for day-to-day operations. Instead, operators can manage network changes through a centralized single pane of glass, enabling them to perform cross-domain tasks—such as provisioning services or implementing security policies—quickly and efficiently. This capability not only accelerates the deployment of changes but also reduces the potential for human error, ensuring consistency and alignment across all domains, including Campus, WAN, and Data Center.

By abstracting the complexity of low-level configurations and enabling operators to work at a service-centric level, northbound integration transforms the network management experience, driving operational agility and scalability while maintaining the integrity of the overall network architecture.

Deployment architecture

The deployment architecture for this customer is built on the Network as Code framework, leveraging the DevOps methodology. The design utilizes a unified Campus, WAN, and Data Center (DC) Data Model. Each Data Model encapsulates the network configuration and policies across all regions, organized into logically separated folders. This hierarchical structure simplifies management and operations while maintaining clarity and ease of use.

Changes made within a Data Model are automatically propagated to all controllers within the same domain. For example, adding a new network segment within the Campus Data Model results in the configuration being synchronized across four separate Campus controllers spanning three regions. This ensures consistency and eliminates the risk of configuration discrepancies across geographically dispersed controllers.

© 2026 Cisco and/or its affiliates. All rights reserved.

Similarly, security policies are deployed globally across all Network Access Control (NAC) engines, ensuring that user access policies remain consistent, regardless of where an endpoint is connected. This uniformity enhances the overall security posture by standardizing access policies and reducing the complexity of managing region-specific configurations.

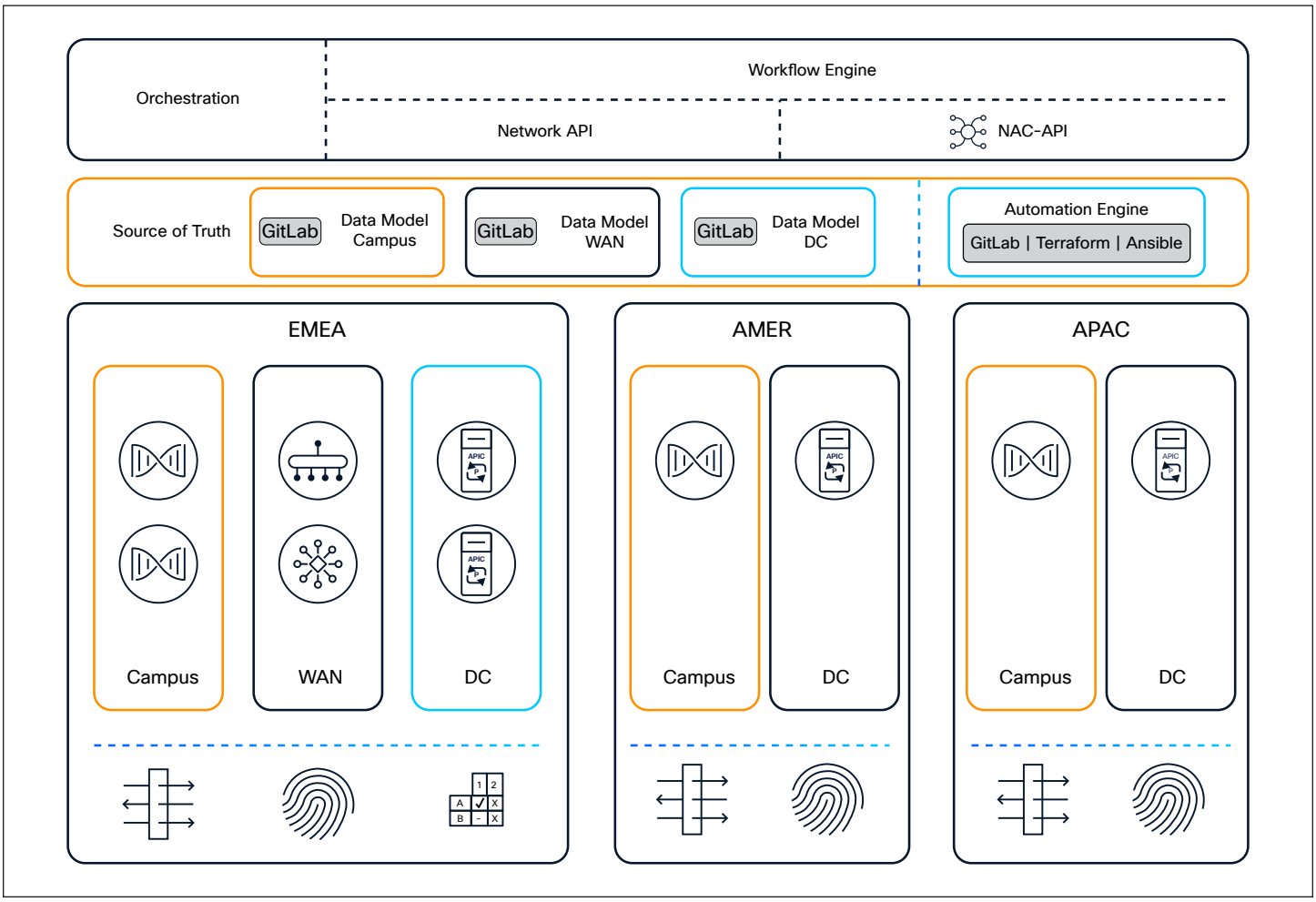


Figure 29. Solution Overview of Multi-Architecture Deployment using Network as Code

Value Proposition and Outcome Achieved

The deployment architecture, built on the Network as Code framework, delivers a transformative approach to managing large-scale, multi-region, and multi-domain networks.

By implementing this architecture, the customer achieved the following tangible outcomes:

- **Improved Efficiency and Accuracy:** Automation eliminates the need for repetitive, manual tasks, reducing errors and ensuring accurate changes across all domains.

© 2026 Cisco and/or its affiliates. All rights reserved.

- **Enhanced Security Posture:** Consistent deployment of network configuration and security policies globally (e.g., user access control, firewall policies) ensures robust protection against vulnerabilities and misconfigurations.
- **Reduced Operational Overhead:** A unified framework minimizes the effort required to manage multiple controllers and regions, freeing up resources for strategic tasks.
- **Streamlined Change Management:** Changes are tested, replicated, and deployed seamlessly across domains, resulting in faster implementation and reduced downtime.
- **Scalability and Future-Readiness:** The architecture supports rapid growth, enabling the addition of new network segments, sites, and applications without disrupting existing operations.
- **Global Standardization:** Uniform policies and configurations provide a consistent user experience and simplify troubleshooting across regions.

Service Provider – End-to-End Data Center Automation

This large Service Provider customer aspired to deploy an automation framework that leverages infrastructure-as-code as an industry standard for their cloud data center infrastructure. They wanted to address the following challenges:

- Having a common automation framework to manage the high number of new infrastructure elements being deployed (new data centers being built) and multiple network changes that occur in operations.
- Translating manual tasks into robust, distributable code.
- Integrating configuration and operational network health checks into the configuration deployment workflow to reduce human errors.
- Relying on industry best practices across all their data center infrastructure elements (version control, automated testing, release tagging, continuous delivery, etc.).
- Supporting the adoption of DevOps practices across their organization.

The Approach

Cisco introduced the Network as Code framework supporting the ACI deployment for data center Networking. The UI and CI/CD functionality are provided by the Continuous Development and Automation Framework (CDAF) platform from Cisco CX, which already exists in the account to support current DevOps processes.

The workflow relies on the user initiating a new config change in a separate Git feature-branch. The human operator reviews and approves the change using the UI offered by CDAF. Once approved, this triggers the CI/CD platform which extracts the required credentials, performs the semantic and syntactic validation of the change, and deploys it leveraging Terraform and the APIC controller.

© 2026 Cisco and/or its affiliates. All rights reserved.

After deployment, the ACI configuration change is validated through a set of 20 tests being run in Cisco CX Test Manager (CXTM), covering operational status, faults, and key configuration consistency checks. The test results are automatically published in a CXTM-driven test report that the operational team reviews before approving the merge of the feature-branch change into the main branch, signalling the successful completion of the operation.

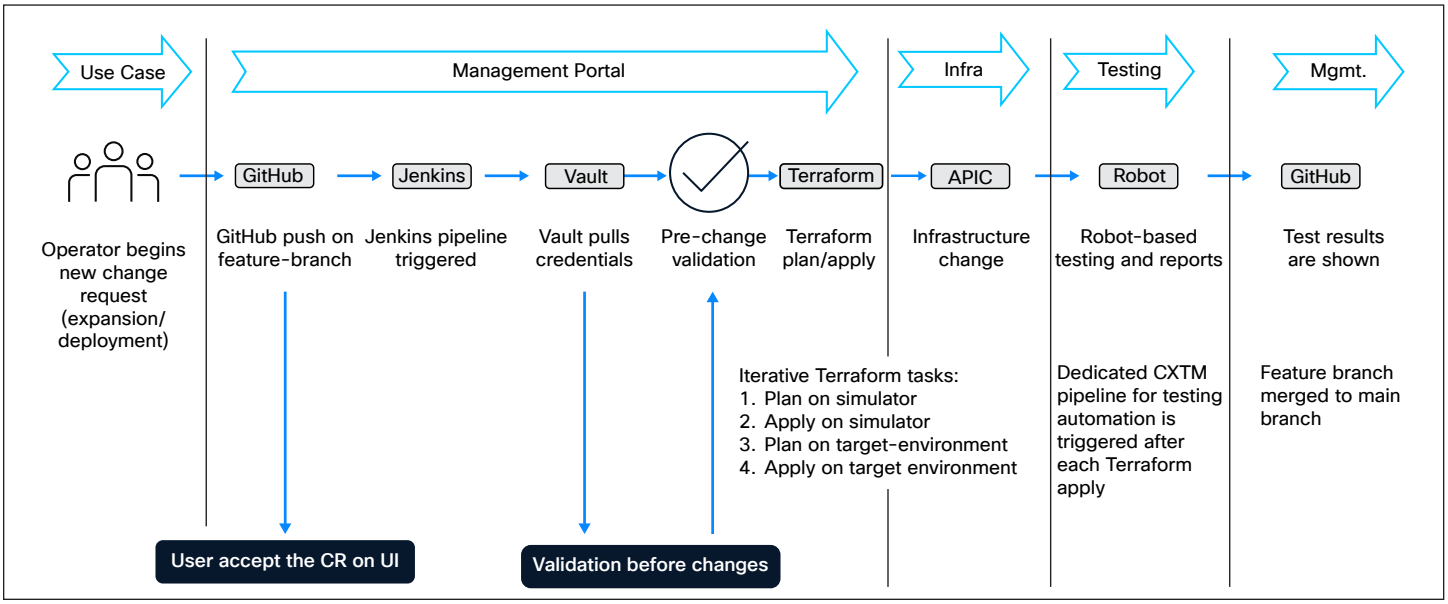


Figure 30. ACI Deployment Automation/Expansion Workflow

Evolution towards End-to-End coverage of the entire data center infrastructure

FFollowing the successful operationalization of ACI as Code, the customer requested that a similar approach be followed for the rest of the data center infrastructure elements: Cisco UCS and VIM, and third-party F5 (used for load balancing), RedHat OpenShift (container hosting platform), and Bacula (backup and restore).

This results in the creation of an End-to-End pipeline from which each individual pipeline element can be executed individually and in sequence, following the workflow presented above.

Each individual product deployment pipeline includes checkout of configuration data, performing pre-checks of the planned configuration change, CI/CD-based deployment, and deployment validation using CXTM-based test automation.

While the ACI as Code element uses a structured data model for configuration modelling and a Terraform module/ provider for deployment, other elements rely on specific API call sequence for configuration and Ansible adaptors for deployment.

© 2026 Cisco and/or its affiliates. All rights reserved.

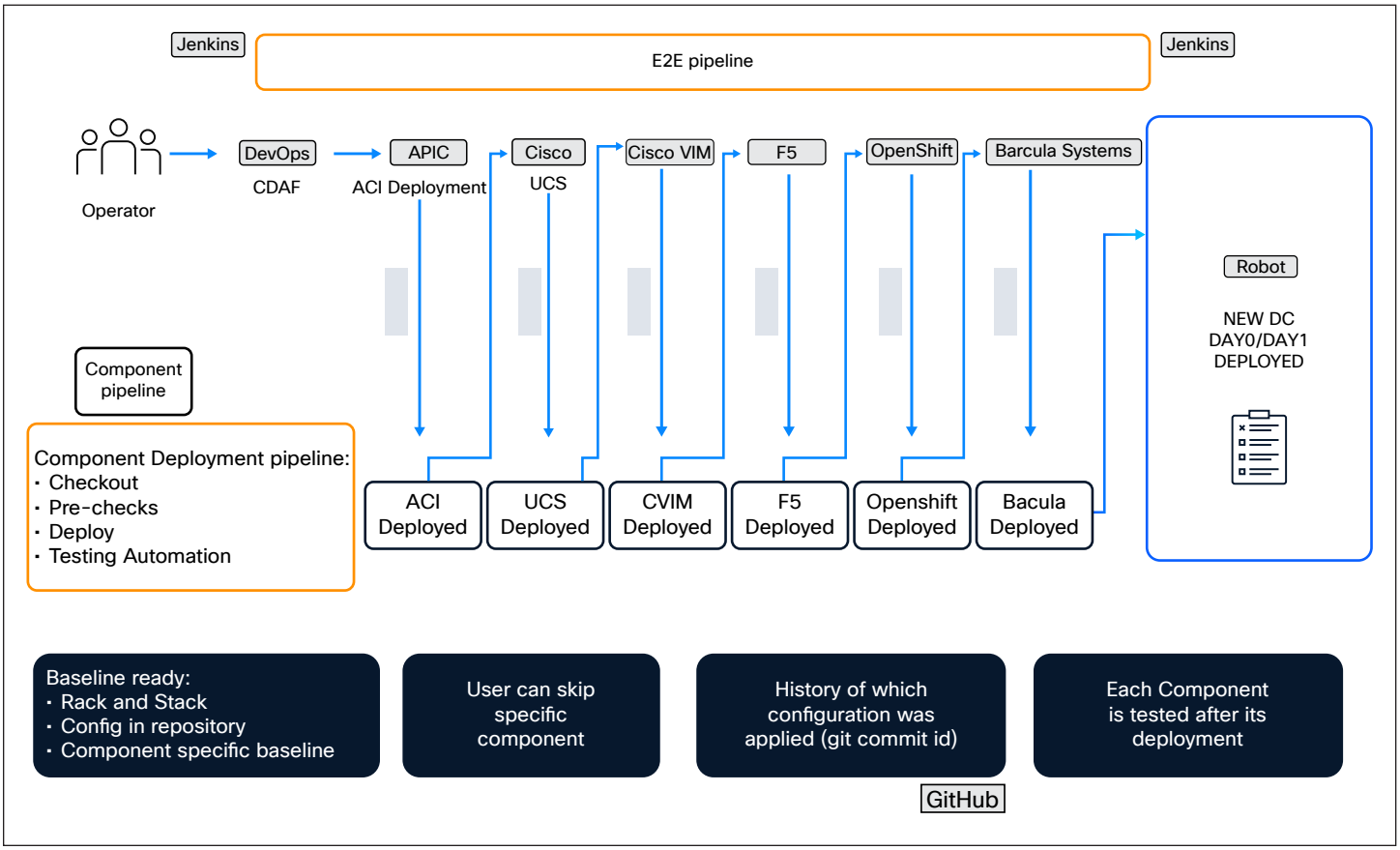


Figure 31. End to End Deployment Automation

Use-cases Delivered

The above framework enables the customer to implement automation use cases covering:

- Complete automation of End-to-End data center deployment configuration.
- Automation of an existing data center expansion—automating the required configuration when a new compute rack must be deployed, including the required data center networking and Cisco Virtual Infrastructure Manager (CVIM)/RedHat Open Compute Project (OCP) platforms.
- Automation of key configuration lifecycle operational tasks such as configuration change lifecycle and seamless re-configuration due to faulty hardware replacement (RMA).

© 2026 Cisco and/or its affiliates. All rights reserved.

Outcome and Business Benefits

The approach above enables the customer to benefit from Cisco Network as Code for ACI, brought by the Service as Code service offering, and expand it to cover the entire cloud infrastructure, including compute, virtualization platforms, and third-party elements such as load-balancers, Kubernetes platforms, and backup/restore.

This approach and the cloud infrastructure expansion resulted in the following business benefits:

- Adoption of a unified DevOps approach covering the entire cloud infrastructure, including third-party products.
- Ensuring there is a seamless adoption of data model–driven configuration when this becomes available for the remainder of the cloud infrastructure elements.
- 50% reduction of a new data center configuration time, with ACI fabric configuration reduced from hours of manual work to 23 minutes, including testing and validation.
- 80% reduction of configuration time for an existing data center expansion.
- Proven value of the data model approach as an enabler for template creation, compliance checks, pre-change validation, and config-drift detection.

Need help getting started? Cisco CX Services have you covered

For organizations or enterprises unfamiliar with the Network as Code or GitOps approach, Cisco CX (Customer Experience) offers a professional service called Services as Code [18] tailored to your organization's needs to ensure a smooth transition and successful adoption. Whether your organization needs guidance, collaboration, or a fully managed solution, Cisco CX can provide support through three flexible engagement models:

- **Do It Yourself:** For teams that prefer to lead the implementation independently, Cisco CX provides expert guidance, best practices, and knowledge transfer. This empowers your organization to confidently adopt Network as Code principles while maintaining full ownership of the process.
- **Do It With You:** Collaborate with Cisco CX experts to co-develop and implement a Network as Code strategy tailored to your organization's needs. This partnership ensures your team gains hands-on experience while leveraging Cisco's expertise to accelerate success, with less risk.
- **Do It for You:** For organizations seeking a fully managed solution, Cisco CX can handle the implementation and ongoing management of Network as Code practices. This enables your team to focus on core business priorities while Cisco ensures optimal performance and efficiency.

Cisco CX professional services provide a unique blend of expertise, tools, and flexibility to meet organizations where they are in their digital transformation journey. Whether you're just starting out or looking to scale, Cisco CX ensures you unlock the full potential of Network as Code for modern network management.

© 2026 Cisco and/or its affiliates. All rights reserved.

Differentiating Cisco CX: Why Cisco?

Cisco stands out as a trusted partner for adopting Network as Code because of its unmatched expertise in networking, automation, and DevOps practices. With industry-leading tools, proven methodologies, and a customer-first mindset, Cisco CX services deliver:

- **Proven Best Practices:** Backed by extensive experience in deploying modern network solutions across industries.
- **Tailored Solutions:** Flexible engagement models designed to fit the specific needs and maturity levels of your organization.
- **Continuous Support:** Ongoing assistance to optimize, scale, and future-proof your network infrastructure.

Cisco CX isn't just a service provider — it's your strategic partner for navigating the complexities of network modernization with confidence and agility.

Summary, Benefits, and Value proposition

In today's fast-paced digital environment, reducing downtime, increasing operational efficiency, and enabling rapid innovation are key to staying competitive. Adopting a Network as Code approach for network management offers transformative benefits by integrating automation, validation, and testing into operational workflows. This methodology leverages DevOps principles and version-controlled repositories to create a more efficient, resilient, and scalable infrastructure.

By shifting to Network as Code, organizations can achieve measurable results, such as reduced operational overhead, faster time to market, and improved network reliability.

Key benefits include:

- **Reduction of Manual Errors:** Automating configuration management and compliance checks ensures more accurate and consistent operations. By minimizing human intervention in routine tasks, organizations can significantly reduce the risk of errors that often lead to downtime, security vulnerabilities, or increased operational costs.
- **Stable and Incremental Changes:** In a Network as Code workflow, all changes to the network are thoroughly tested and applied incrementally. This ensures stability and significantly reduces risks associated with large, untested deployments. Incremental updates also minimize downtime, enabling seamless and uninterrupted network operations..
- **Enhanced Compliance and Traceability:** Network as Code, leveraging GitOps principles, uses version-controlled repositories as the single source of truth. This provides full visibility into who made changes, what was changed, and when it was implemented. Such traceability not only simplifies compliance with regulatory requirements but also makes audits more efficient and less time-consuming.
- **Accelerated Deployment and Agility:** With automation at its core, Network as Code streamlines deployment processes, enabling faster delivery of updates and configurations. This improved agility allows organizations to respond more quickly to evolving business needs, technological changes, and competitive pressures, reducing time to market for critical services or products.

© 2026 Cisco and/or its affiliates. All rights reserved.

- **Improved Collaboration and Consistency:** By centralizing configurations in Git-based repositories, teams can collaborate more effectively through code reviews and pull requests. This fosters a culture of shared responsibility and ensures consistent configurations across environments. Cross-functional teams, including network engineers, developers, and operations staff, can work together more seamlessly.
- **Risk Mitigation and Easy Rollbacks:** The ability to quickly roll back to a previously stable configuration in the event of an issue reduces downtime and mitigates risks associated with deploying new changes. This capability ensures business continuity and minimizes the impact of unexpected errors.
- **Seamless Integration with Existing Infrastructure as Code (IaC):** Network as Code complements IaC practices by treating network infrastructure configurations like software code. This alignment simplifies management, supports rapid scaling, and aligns with modern development workflows, ensuring that the network evolves alongside the organization's broader IT strategy.

When implementing Network as Code, organizations can achieve greater operational efficiency, reduce risks, and adapt to change with confidence. This solution not only ensures a more stable and predictable infrastructure, but also accelerates time to value, fosters collaboration across teams, and empowers organizations to innovate at speed.

Business Impact and ROI: Unlocking Competitive Advantage

Organizations that adopted Network as Code often experience measurable improvements in key performance areas such as:

- **98%+ Configuration Change Success:** Automation reduces human intervention, minimizing mistakes and the operational overhead required to fix them..
- **5x Faster Deployment Times:** Streamlined workflows and pre-tested updates accelerate time to market for new services and capabilities.
- **Improved Reliability and Uptime:** Incremental and validated changes reduce downtime, ensuring critical business operations remain uninterrupted.

Modernizing network configuration management enables organizations to unlock significant cost savings, improve service delivery, and strengthen their competitive position in the market.

© 2026 Cisco and/or its affiliates. All rights reserved.

Authors

Morten Skriver, Principal Architect, Cisco Customer Experience

Ankush Arora, Distinguished Architect, Cisco Customer Experience

Justyna Chowaniec, Customer Delivery Architect, Cisco Customer Experience

Tomasz Zarski, Customer Delivery Architect, Cisco Customer Experience

Aditya Nath Singh, Customer Delivery Software Architect, Cisco Customer Experience

Krishna Bandi, Customer Delivery Engineering Technical Leader, Cisco Customer Experience

Senthil Kumar, Customer Delivery Architect, Cisco Customer Experience

Stefan-Alexandru Manza, Distinguished Architect, Cisco Customer Experience

Robert Crowe, Principal Architect, Cisco Customer Experience

Andrea Testino, Principal Software Engineer, Cisco Customer Experience

Rafael Muller, Principal Engineer, Cisco Customer Experience

Vladimir Joshevski, Product Management Architect, Cisco Customer Experience

Marcin Hamróz, Principal Architect, Cisco Customer Experience

Reviewers

Michael Kaemper, CTO EMEA, Cisco Customer Experience

Asier Arlegui, Principal Architect, Cisco Customer Experience

Greg Leider, Product Leader, Cisco Customer Experience

© 2026 Cisco and/or its affiliates. All rights reserved.

Citations and bibliography

1. Cisco, How automation is driving network engineer skills transformation, https://www.cisco.com/c/dam/en/us/solutions/collateral/executive-perspectives/technology-perspectives/automation-driving-network-eng-skills-trans.pdf.
2. Cisco, Single Source of Truth in Network Automation White Paper, https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/technology-perspectives/ssot-nw-automation-wp.html.
3. Cisco, Cisco Application Centric Infrastructure (ACI), https://www.cisco.com/site/us/en/products/networking/cloud-networking/application-centric-infrastructure/index.html.
4. OpenConfig, OpenConfig, https://www.openconfig.net.
5. 23andMe, Yamale, https://github.com/23andMe/Yamale.
6. Cisco, Cisco SD-WAN Manager, https://www.cisco.com/site/us/en/products/networking/wan/vmanage/index.html.
7. Cisco, Cisco Meraki, https://meraki.cisco.com/products/meraki-dashboard/.
8. RedHat, Ansible, https://www.ansible.com/.
9. HashiCorp, Terraform, https://www.terraform.io.
10. L. Foundation, Open Tofu, https://opentofu.org.
11. Cisco, Nexus Dashboard Fabric Controller (NDFC), https://www.cisco.com/c/en_uk/products/cloud-systems-management/prime-data-center-network-manager/index.html.
12. Cisco, nac-validate, https://github.com/netascode/nac-validate.
13. Cisco, nac-test, https://github.com/netascode/nac-test.
14. HashiCorp, Valult, https://www.hashicorp.com/en/products/vault.
15. HashiCorp, Terraform Workspaces, https://developer.hashicorp.com/terraform/language/state/workspaces.
16. HashiCorp, Terraform Backend Configuration, https://developer.hashicorp.com/terraform/language/backend.
17. Cisco, Scaling Network as Code, https://netascode.cisco.com/docs/guides/deployment/scaling/.
18. Cisco, Services as Code, https://www.cisco.com/c/en/us/services/collateral/se/services-as-code-aag.html.

© 2026 Cisco and/or its affiliates. All rights reserved. Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: www.cisco.com/go/trademarks. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)

C11-5823739-00 03/26