

CISCO *Live!*

Let's go



The bridge to possible

# Nexus-as-Code

Kickstart your automation with ACI

Daniel Schmidt, Principal Architect

CISCO *Live!*

BRKDCN-2673

# Agenda

- Infrastructure as Code
- Introduction to Nexus-as-Code
- Pre-Change Validation
- Automated Testing
- CI/CD Integration
- Services as Code

# Infrastructure as Code



WIKIPEDIA  
The Free Encyclopedia

Infrastructure as code (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.



Infrastructure as Code (IaC) is the management of infrastructure in a descriptive model, using the same versioning as DevOps team uses for source code.



Infrastructure as Code (IaC) is the managing and provisioning of infrastructure through code instead of through manual processes.

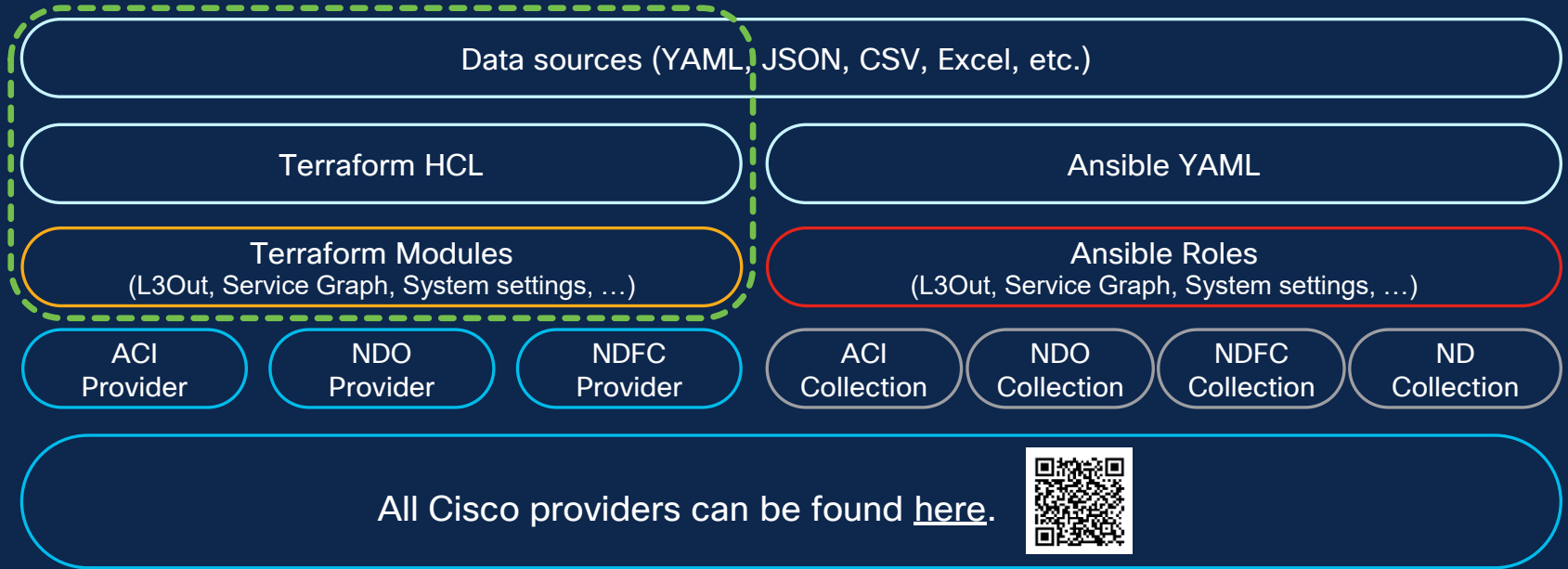


Practicing infrastructure as code means applying the same rigor of application code development to infrastructure provisioning. All configurations should be defined in a declarative way and stored in a source control system.

**Infrastructure as Code is a process, not a single tool or application**

# IaC Strategy

## Nexus-as-Code



# Terraform Primer

## Terraform is an Infrastructure Resources Manager

- Compose and combine infrastructure resources to build and maintain a desired state
- Plan and execution are distinct actions
- Manages all resources through APIs
- Terraform uses core and plugin components for basic functions and extensibility
- One of the most used IaC (Infrastructure-as-Code) tools to manage public Cloud and Datacenter assets
- HCL (Terraforms underlying configuration language) is the fastest growing language on GitHub in 2022 \*



```
provider "aci" {
  username = "admin"
  password = "Cisco123"
  url      = "https://10.1.1.1"
}

resource "aci_vlan_pool" "VP1" {
  name       = "VP1"
  alloc_mode = "static"
}

resource "aci_ranges" "RANGE1" {
  vlan_pool_dn = aci_vlan_pool.VP1.dn
  from         = 1000
  to           = 1099
}
```

\* <https://octoverse.github.com/2022/top-programming-languages>

# Nexus-as-Code

<https://cisco.com/go/nexusascode>



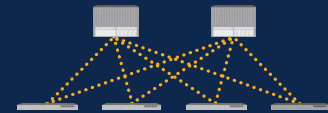
- Nexus-as-Code aims to reduce time to value by lowering the barrier of entry to network orchestration through simplification, abstraction, and curated examples.
- It allows users to instantiate network fabrics in minutes using an easy to use, opinionated data model. It takes away the complexity of having to deal with references, dependencies or loops.
- Users can focus on describing the intended configuration while using a set of maintained and tested Terraform Modules without the need to understand the low-level ACI object model.

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



HashiCorp

**Terraform**



# Comparison

## Native Terraform

```
resource "aci_tenant" "tenant_CiscoLive" {
  name = "CiscoLive"
}

variable "vrfs" {
  default = {
    VRF1 = {
      name = "VRF1"
    },
    VRF2 = {
      name = "VRF2"
    }
  }
}

resource "aci_vrf" "vrfs" {
  for_each = var.vrfs
  tenant_dn = aci_tenant.tenant_CiscoLive.id
  name      = each.value.name
}
```

## Nexus-as-Code

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



# Node Policies

- The data model is organized in a way that configurations are grouped around where the actual configuration (policy) is applied.
- All the configurations that are applied at the node level can be found under:  
`apic -> node_policies -> nodes`
- This includes configurations typically found in different places in the ACI object tree, like for example the OOB node management address, which is configured under the `mgmt` tenant.
- Consolidating all node level configurations in a single place eases maintenance, as for example we only have to update this single section when adding a new node.

```
apic:
  node_policies:
    nodes:
      - id: 101
        pod: 2
        role: leaf
        serial_number: FDO13026BEN
        name: leaf-101
        oob_address: 10.103.5.101/24
        oob_gateway: 10.103.5.254
        update_group: group-1
        fabric_policy_group: all-leafs
        access_policy_group: all-leafs

      - id: 1
        pod: 2
        role: apic
        oob_address: 10.103.5.1/24
        oob_gateway: 10.103.5.254
```

# Access Policies

- A number of profiles and selectors can be auto-generated by providing a naming convention.
- There is no need to worry about any of the profiles and selectors as they will be added/deleted automatically according to the node and interface configuration.
- As nodes are added under `apic -> node_policies -> nodes` the corresponding profiles will be created automatically.
- Once interface configurations are added under `apic -> interface_policies -> nodes -> interfaces` the corresponding interface selectors will be created.

```
apic:
  auto_generate_switch_pod_profiles: true

interface_policies:
  nodes:
    - id: 101
      interfaces:
        - port: 1
          description: Linux Server 1
          policy_group: linux-servers
        - port: 2
          description: Linux Server 2
          policy_group: linux-servers
        - port: 47
          description: N7K Core
          policy_group: n7000-a
        - port: 48
          description: N7K Core
          policy_group: n7000-b
```

# Simple Demo

<https://github.com/netascode/nac-aci-simple-example>



# Separate Data from Code

In order to ease maintenance we separate data (variable definition) from logic (infrastructure declaration), where one can be updated independently from the other.

```
apic:
  tenants:
    - name: CiscoLive
      vrfs:
        - name: CiscoLive
      bridge_domains:
        - name: vlan-100
          vrf: CiscoLive
      application_profiles:
        - name: dev
          endpoint_groups:
            - name: vlan-100
              bridge_domain: vlan-100
              physical_domains: ["12"]
```

 apic.yaml

```
module "aci" {
  source = "netascode/nac-aci/aci"
  version = "0.8.1"

  yaml_directories = ["data"]

  manage_access_policies      = true
  manage_fabric_policies      = true
  manage_pod_policies         = true
  manage_node_policies        = true
  manage_interface_policies   = true
  manage_tenants              = true
}
```

 main.tf

# ACI Terraform Provider

- Nexus-as-Code heavily relies on the generic `aci_rest_managed` resource of the ACI Terraform provider.
- This fully-featured resource is able to manage any ACI object.
- The resource is not only capable of pushing a configuration but also reading its state and reconcile configuration drift.

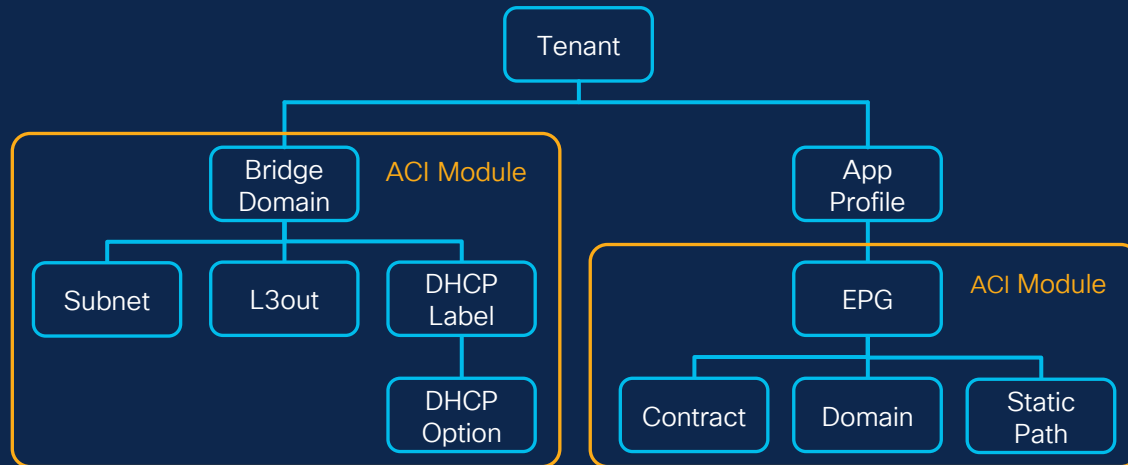
```
resource "aci_rest_managed" "fvTenant" {
  dn          = "uni/tn-EXAMPLE_TENANT"
  class_name = "fvTenant"

  content = {
    name = "EXAMPLE_TENANT"
    descr = "Example description"
  }

  child {
    rn          = "ctx-VRF1"
    class_name = "fvCtx"
    content = {
      name = "VRF1"
    }
  }
}
```

# ACI Modules

- Terraform Modules allow us to introduce a level of abstraction similar to functions in programming languages
- Where a Terraform resource typically represents a single ACI object, a Terraform module can represent a branch in the object tree



# ACI Module Example

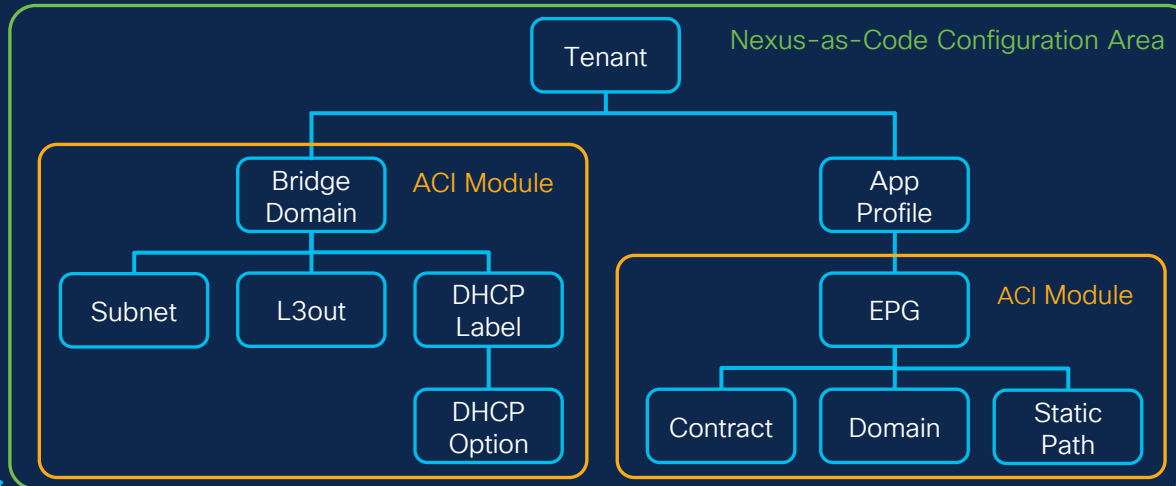
- Modules allow us to break a configuration into more manageable pieces which can be developed and tested independently
- Modules can be versioned and released independently
- Modules enable easier shareability and cut down on duplicate work as they can be shared with the wider community (Terraform Registry)
- The Terraform Registry allows publishing a module and an optional set of sub-modules from a single repository

```
module "aci_endpoint_group" {
  source = "netascode/nac-aci/aci//modules/terraform-aci-endpoint-group"
  version = "0.8.1"

  tenant           = "ABC"
  application_profile = "AP1"
  name             = "EPG1"
  bridge_domain    = "BD1"
  contract_consumers = ["CON1"]
  physical_domains = ["PHY1"]
  vmware_vmm_domains = [{
    name = "VMW1"
  }]
  static_ports = [{
    node_id = 101
    vlan = 123
    port = 10
  }]
}
```

# Nexus-as-Code Module

- **Fabric Policies:** Configurations applied at the fabric level (e.g., fabric 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., OOB node management address)
- **Interface Policies:** Configurations applied at the interface level (e.g., assigning interface policy groups to ports)
- **Tenants:** Configurations applied at the tenant level (e.g., VRFs and Bridge Domains)





# YAML Layout

- As different teams might be responsible for different parts of the infrastructure, it is of paramount importance to allow enough flexibility when defining and maintaining the ACI configuration.
- The configuration can be split into multiple YAML files each for example covering a specific logical section of the configuration.
- Nexus-as-Code does not dictate a specific schema, but instead allows for full flexibility to divide the configuration as needed.

```
$ tree -L 2
.
├── data
│   ├── apic.yaml
│   ├── access_policies.yaml
│   ├── fabric_policies.yaml
│   ├── node_policies.yaml
│   ├── pod_policies.yaml
│   ├── node_1001.yaml
│   ├── node_101.yaml
│   ├── node_102.yaml
│   ├── tenant_PROD.yaml
│   └── defaults.yaml
└── main.tf
```

# Deep Merge YAML Content

YAML files can be split at arbitrary points, meaning the Nexus-as-Code Module will combine and deep merge the contents of YAML files, where data of two elements with the same keys will be combined. This for example enables splitting the configuration of a single tenant in two YAML files.



Management Service

```
apic:
  tenants:
    - name: PROD
      vrfs:
        - name: MANAGEMENT
      bridge_domains:
        - name: VLAN100
          vrf: MANAGEMENT
```



HR Service

```
apic:
  tenants:
    - name: PROD
      vrfs:
        - name: HR
      bridge_domains:
        - name: VLAN200
          vrf: HR
```

# Sensitive Information

The configuration might contain sensitive information that should not be stored in cleartext in the configuration. One common approach to handling secrets in the context of CI/CD Platforms is by injecting sensitive values as environment variables during runtime.

```
apic:  
  access_policies:  
    mcp:  
      key: !env MCP_KEY
```



CI/CD Platform



Secrets Storage

```
MCP_KEY: Cisco123Cisco123
```

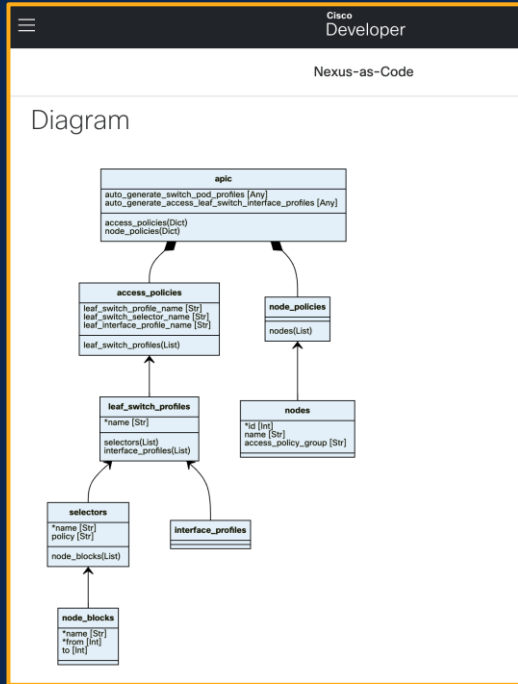


Runtime Environment

```
apic:  
  access_policies:  
    mcp:  
      key: Cisco123Cisco123
```

# Data Model Documentation

<https://cisco.com/go/nexusascode>



Cisco Developer

Nexus-as-Code

nodes (*apic.node\_policies*)

Name	Type	Constraint	Mandatory	Default
id	Integer	min: 1, max: 4000	Yes	
name	String	Regex: <code>^[a-zA-Z0-9_!-]{1,64}\$</code>	No	
access_policy_group	String	Regex: <code>^[a-zA-Z0-9_!-]{1,64}\$</code>	No	

Cisco Developer

Nexus-as-Code

YAML

```
apic:
  access_policies:
    leaf_switch_profiles:
      - name: LEAF1001
        selectors:
          - name: SEL1
            policy: ALL_LEAFS
            node_blocks:
              - name: BLOCK1
                from: 1001
            interface_profiles:
              - LEAF1001
```

Copy

Cisco Developer

Nexus-as-Code

## Access Leaf Switch Profile

Leaf Switch Profiles can either be auto-generated, one per leaf, by providing a naming convention or can be defined explicitly. In case of auto-generated profiles the following placeholders can be used when defining the naming convention:

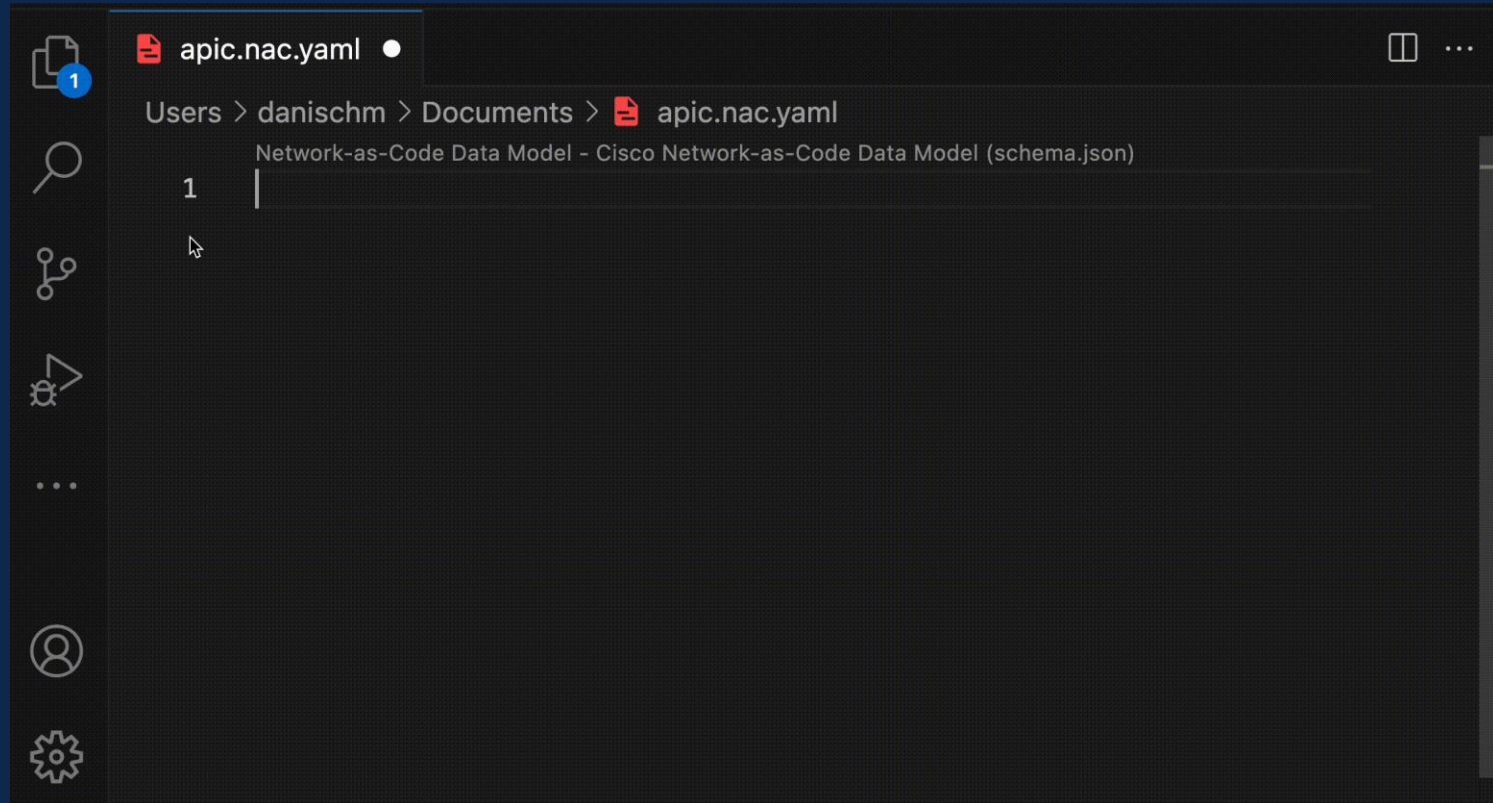
- `<id>`: gets replaced by the respective leaf node ID
- `<name>`: gets replaced by the respective leaf hostname

Location in GUI:  
Fabric » Access Policies » Switches » Leaf Switches » Profiles

### Terraform modules

- [Access Leaf Switch Profile](#)

# Visual Studio Code Integration



# Default Values

- Nexus-as-Code comes with pre-defined default values based on common best practices.
- In some cases, those default values might not be the best choice for a particular deployment and can be overwritten if needed.
- Appending suffixes to object names is a common practice that introduces room for human errors. Using default values, such suffixes can be defined once and then consistently appended to all objects of a specific type including its references.

```
defaults:  
  apic:  
    tenants:  
      bridge_domains:  
        name_suffix: _bd  
        unicast_routing: false
```



defaults.yaml

```
apic:  
  tenants:  
    - name: CiscoLive  
      bridge_domains:  
        - name: vlan_101  
        - name: vlan_102  
        - name: vlan_103
```



tenants.yaml

# Unmanaged Parent Objects

In some cases you might only want to manage objects within a container. The `managed` flag indicates if an object should be created/modified/deleted or is assumed to exist already and just acts a container for other objects.



Infrastructure Team manages Tenants

```
apic:
  tenants:
    - name: Dev
    - name: Stage
    - name: Prod
```



Developers manage Tenant Objects

```
apic:
  tenants:
    - name: Dev
      managed: false
      vrfs:
        - name: VRF1
        - name: VRF2
```

# Incorporate Data from Other Sources



## VLANs

Name	ID
VLAN101	101
VLAN102	102

```
apic:
  tenants:
    - name: PROD
  vrfs:
    - name: PROD
```

```
data "netbox_vlans" "nbv" {
}

locals {
  model = {apic = {tenants = [{
    name = "PROD"
    bridge_domains = [for vlan in data.netbox_vlans.nbv.vlans : {
      name = vlan.name
      vrf = "PROD"
    }]
  ]}]}}
}

module "aci" {
  source = "netascode/nac-aci/aci"
  version = "0.8.1"

  yml_directories = ["data"]
  model            = local.model
  ...
}
```



# Pre-Change Validation

As the complexity of the configuration and the underlying data model increases automated validation before deploying anything in a production environment becomes a critical aspect.

Several tools can be used to ensure that the provided input data is valid, but also that common best practices and formatting guidelines are being followed.



# Pre-Change Validation



iac-validate



A CLI tool to perform format, syntactic, semantic and compliance validation of Nexus-as-Code YAML files.

```
$ iac-validate -h
Usage: iac-validate [OPTIONS] [PATHS]...

A CLI tool to perform syntactic and semantic validation of YAML files.

Options:
  --version           Show the version and exit.
  -v, --verbosity LVL Either CRITICAL, ERROR, WARNING, INFO or DEBUG
  -s, --schema FILE  Path to schema file. (optional, default:
                    '.schema.yaml', env: IAC_VALIDATE_SCHEMA)
  -r, --rules DIRECTORY Path to semantic rules. (optional, default:
                    '.rules/', env: IAC_VALIDATE_RULES)
  -o, --output FILE  Write merged content from YAML files to a new YAML
                    file. (optional, env: IAC_VALIDATE_OUTPUT)
  -h, --help         Show this message and exit.
```

# Syntax Validation



iac-validate



- Native Terraform variable validation rules have limitations with complex and/or nested structures
- Tools like **Yamale** can be used to define the schema and validate YAML files against it
- The schema specifies the expected structure, input value types (String, Enum, IP, etc.) and additional constraints (eg. value ranges, regexes, etc.)

```
---
apic: include('apic', required=False)
---
apic:
  tenants: list(include('tenant'), required=False)

tenant:
  name: regex('^[a-zA-Z0-9_.-]{1,64}$')
  vrfs: list(include('ten_vrf'), required=False)

ten_vrf:
  name: regex('^[a-zA-Z0-9_.-]{1,64}$')
  alias: regex('^[a-zA-Z0-9_.-]{1,64}$', required=False)
  data_plane_learning: bool(required=False)
  enforcement_direction: bool(required=False)
  contracts: include('ten_vrf_contracts', required=False)
```

# Semantic Validation



iac-validate



Semantic validation is about verifying specific data model related constraints like referential integrity. It can be implemented using a rule based model like commonly done with linting tools. Examples are:

- Check uniqueness of key values (eg. Node IDs)
- Check references/relationships between objects (eg. Interface Policy Group referencing a CDP Policy)

```
Rule 101: Verify unique keys ['apic.node_policies.nodes.id - 102']
Rule 201: Verify references ['apic.node_policies.nodes.update_group - GROUP1']
Rule 205: Verify Access Spine Interface Policy Group references
['apic.interface_policies.nodes.interfaces.policy_group - SERVER1']
```

# Compliance Validation

## NDI Pre-Change Analysis

Nexus Dashboard Insights (NDI) is continuously pulling the entire policy, every configuration, and the network-wide state, along with the operator intent, and building from these comprehensive and mathematically accurate models of network behavior. It combines this with codified Cisco domain knowledge to generate “smart events” that pinpoint deviations from intent and offer remediation recommendations.

The Pre-Change Analysis feature can be used to assess the impact of a particular change before applying it to the infrastructure. This is done by applying the planned changes to the model and then analysing the impact.



# NDI Pre-Change Validation



nexus-pcv



A CLI tool to perform a pre-change analysis on Nexus Dashboard Insights or Network Assurance Engine. It can either work with provided JSON file(s) or a [terraform plan](#) output from a Nexus-as-Code project. It waits for the analysis to complete and evaluates the results.

```
$ nexus-pcv -h
Usage: nexus-pcv [OPTIONS]

A CLI tool to perform a pre-change validation on Nexus Dashboard Insights or
Network Assurance Engine.

Options:
  -i, --hostname-ip TEXT      NAE/ND hostname or IP (required, env:
                              PCV_HOSTNAME_IP).
  -u, --username TEXT        NAE/ND username (required, env: PCV_USERNAME).
  -p, --password TEXT        NAE/ND password (required, env: PCV_PASSWORD).
  -d, --domain TEXT          NAE/ND login domain (optional, default: 'Local',
                              env: PCV_DOMAIN).
  -g, --group TEXT           NAE assurance group name or NDI insights group
                              name (required, env: PCV_GROUP).
  -s, --site TEXT            NDI site or fabric name (optional, only required
                              for NDI, env: PCV_SITE).
```

# Testing

There are certain aspects we can only verify after deployment like for example operational state. Various testing frameworks can be used for that, one example would be [Robot Framework](#). Robot's language agnostic syntax with libraries like [Requests](#) and [JSONLibrary](#) can be used to write tests against REST APIs.

In combination with templating languages like Jinja we can render test cases dynamically based on the desired state.

Tests can typically be categorized in three groups:

- **Configuration Tests**: verify if the desired configuration is in place
- **Health Tests**: leverage the in-built APIC fault correlation to retrieve faults and health scores and compare them against thresholds and/or previous state
- **Operational Tests**: verify operational state according to input data, eg. BGP peering state

# Testing



iac-test



A CLI tool to render and execute Robot Framework tests using Jinja templating.

```
$ iac-test -h
Usage: iac-test [OPTIONS]

A CLI tool to render and execute Robot Framework tests using Jinja
templating.

Options:
  -d, --data PATH          Path to data YAML files. (env: IAC_TEST_DATA)
                           [required]
  -t, --templates DIRECTORY Path to test templates. (env: IAC_TEST_TEMPLATES)
                           [required]
  -f, --filters DIRECTORY  Path to Jinja filters. (env: IAC_TEST_FILTERS)
  --tests DIRECTORY        Path to Jinja tests. (env: IAC_TEST_TESTS)
  -o, --output DIRECTORY   Path to output directory. (env: IAC_TEST_OUTPUT)
                           [required]
  -i, --include TEXT       Selects the test cases by tag (include). (env:
                           IAC_TEST_INCLUDE)
  -e, --exclude TEXT       Selects the test cases by tag (exclude). (env:
                           IAC_TEST_EXCLUDE)
  --render-only            Only render tests without executing them. (env:
                           IAC_TEST_RENDER_ONLY)
```



# Robot/Jinja Example



iac-test



```
*** Settings ***
Documentation    Verify Tenant Health
Suite Setup     Login APIC
Default Tags    apic    day2    health    tenants    non-critical
Resource        ../../apic_common.resource

*** Test Cases ***
{% for tenant in apic.tenants | default([]) %}
Verify Tenant {{ tenant.name }} Faults
    ${r}=    GET On Session    apic    /api/mo/uni/tn-{{ tenant.name }}/fltCnts.json
    ${critical}=    Get Value From Json    ${r.json()}    $..faultCountsWithDetails.attributes.crit
    Run Keyword If    ${critical} > 0    Run Keyword And Continue On Failure
    ...    Fail "{{ tenant.name }} has ${critical} critical faults"

Verify Tenant {{ tenant.name }} Health
    ${r}=    GET On Session    apic    /api/mo/uni/tn-{{ tenant.name }}/health.json
    ${health}=    Get Value From Json    ${r.json()}    $..healthInst.attributes.cur
    Run Keyword If    ${health} < 100    Run Keyword And Continue On Failure
    ...    Fail "{{ tenant.name }} health score: ${health}"

{% endfor %}
```

# CI/CD Workflow Example



1

Dev Branch



Push to  
feature branch



Workflow  
triggered



Syntax and  
Semantic Validation



Webex  
notification

2

Pull Request



Open GitHub  
Pull Request



Workflow  
triggered



Terraform  
Plan



NDI Pre-Change  
Analysis



Webex  
notification

3

Main Branch



Merge into  
main branch



Workflow  
triggered



Deployment



Testing



Run NDI Delta  
Analysis



Webex  
notification

# CI/CD Demo

<https://github.com/netascode/BRKDCN-2673-Demo>



# NDO Support

Support for NDO (version 3.7 and 4.2) was recently added to Nexus-as-Code.

```
ndo:
  sites:
    - name: PARIS
      id: 1
      apic_urls: [https://10.1.1.1:443]
  tenants:
    - name: NET
      sites:
        - name: PARIS
  schemas:
    - name: NET
      templates:
        - name: SHARED
          tenant: NET
          vrfs:
            - name: PROD
              sites: [PARIS]
```



ndo.yaml

```
module "ndo" {
  source = "netascode/nac-ndo/mso"
  version = "0.8.1"

  yaml_directories = ["data"]

  manage_system           = true
  manage_sites            = true
  manage_site_connectivity = true
  manage_tenants          = true
  manage_schemas         = true
  deploy_templates        = true
}
```



main.tf

# Scalability

By adding more and more objects to your configuration a few problems can arise:

- The Terraform state file becomes bigger and making changes with Terraform takes much longer.
- A single shared statefile 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 statefile is locked during the operation.
- With Nexus-as-Code, state can be split into multiple workspaces while retaining a single set of YAML files.

```
$ tree -L 2
.
├── data
│   ├── apic.yaml
│   ├── access_policies.yaml
│   ├── fabric_policies.yaml
│   ├── node_policies.yaml
│   ├── pod_policies.yaml
│   ├── node_1001.yaml
│   ├── node_101.yaml
│   ├── node_102.yaml
│   ├── tenant_PROD.yaml
│   ├── tenant_DEV.yaml
│   └── defaults.yaml
└── workspaces
    ├── tenant_PROD
    │   └── main.tf
    └── tenant_DEV
        └── main.tf
```

# Services as Code – Cisco Lifecycle Services

Services as Code is available through Cisco Lifecycle Services as an annual subscription service.

- Readiness **assessment**
- People, process, and solutions **enablement**
- Solution set-up and continuous **integration**
- Comprehensive library of validation rules and **automated test cases**
- Customized development of **new features** and test cases
- Quarterly Business Review **reports**
- Ongoing 24x7 **technical support**

Currently available

ACI



SDWAN



ISE



Firewall



Currently being developed

Catalyst Center



NDFC VXLAN



IP Transport



Meraki



Shifting your infrastructure and operations strategies to focus on driving business outcomes with automation.

# References



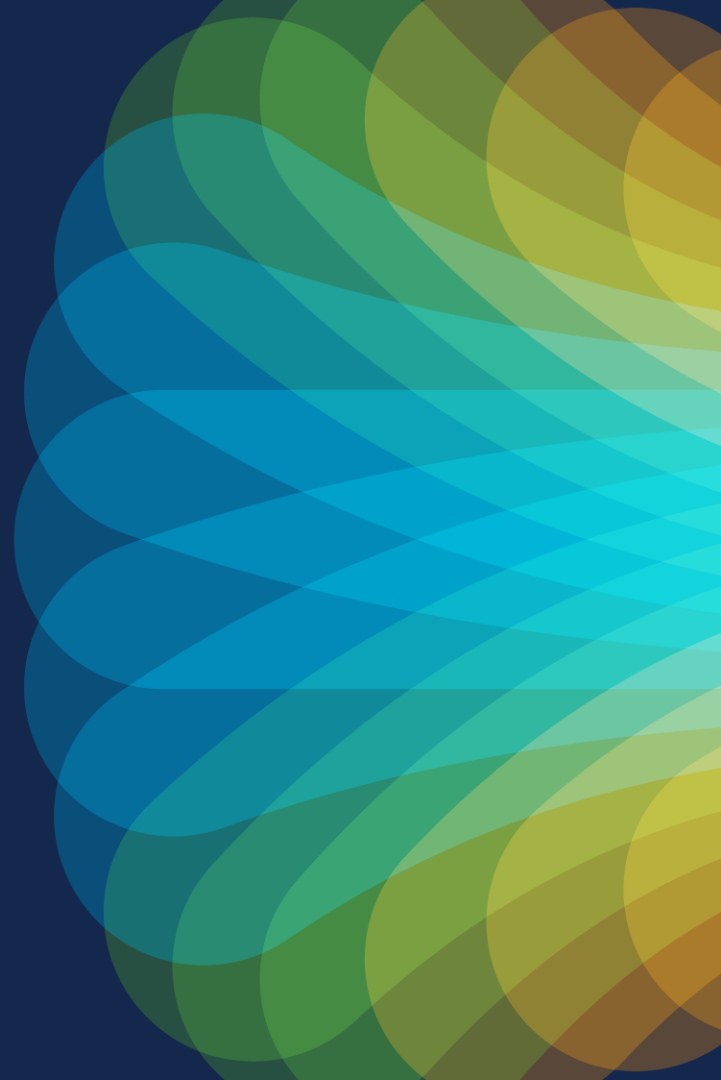
- Nexus-as-Code  
<https://cisco.com/go/nexusascode>
- Demo Repository  
<https://github.com/netascode/BRKDCN-2673-Demo>
- Cisco Lifecycle Service – Services as Code  
<https://www.cisco.com/site/us/en/services/lifecycle-services/index.html>
- Pre-Change Validation Tool  
<https://github.com/netascode/iac-validate>
- Test Automation Tool  
<https://github.com/netascode/iac-test>
- SDWAN, Catalyst Center, ISE, NX-OS, IOS-XE, IOS-XR Terraform Providers  
<https://registry.terraform.io/search/providers?q=CiscoDevNet>



The bridge to possible

# Thank you

CISCO *Live!*





CISCO *Live!*

Let's go