



The bridge to possible

# Lessons Learned from Fully Automated Nexus-as-Code Deployments

Daniel Schmidt, CX Architect

# Cisco Webex App

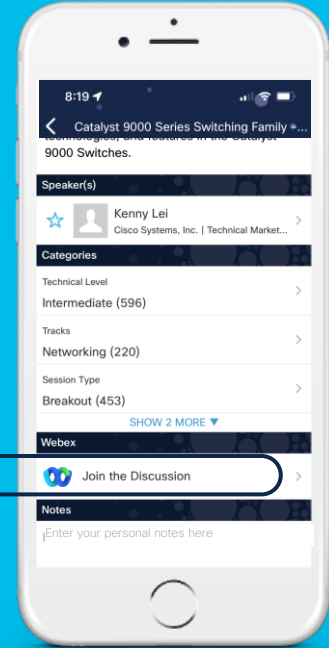
## Questions?

Use Cisco Webex App to chat with the speaker after the session

## How

- 1 Find this session in the Cisco Live Mobile App
- 2 Click “Join the Discussion”
- 3 Install the Webex App or go directly to the Webex space
- 4 Enter messages/questions in the Webex space

Webex spaces will be moderated until February 24, 2023.





# Agenda

- Introduction
- Scaling Terraform Projects
- Pre-Change Validation
- Automated Testing
- CI/CD Integration
- Demo
- Lessons Learned

# Infrastructure as Code



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

# 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>

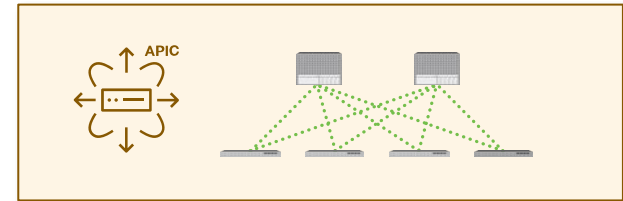
# Nexus-as-Code

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



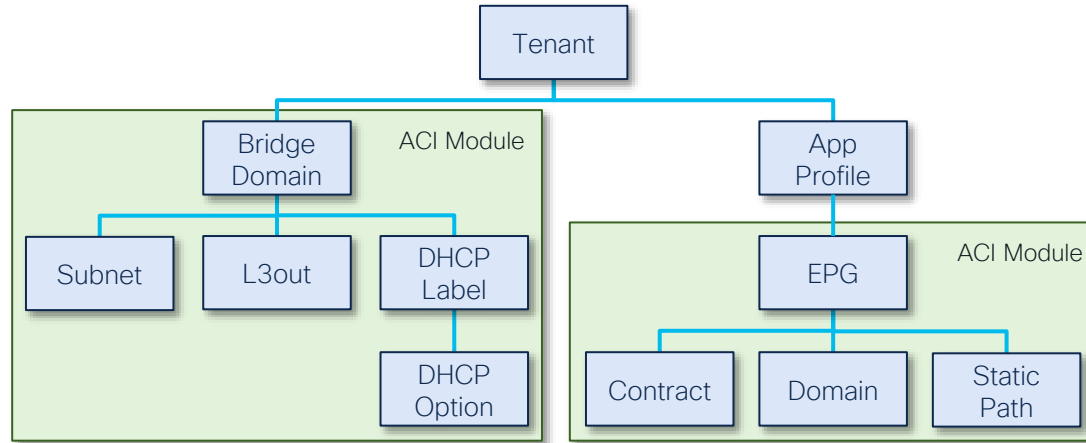
- Nexus-as-Code aims to reduce time to value by lowering the barrier of entry to network orchestration through simplification, abstraction, and curated examples.
- It allows users to instantiate network fabrics in minutes using an easy to use, opinionated data model. It takes away the complexity of having 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
```



# Terraform 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



# Terraform 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)
- Terraform recently introduced a testing experiment, which enables writing integration tests for modules directly in Terraform

```
module "aci_endpoint_group" {  
  source = "netascode/endpoint-group/aci"  
  version = ">= 0.1.0"  
  
  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  
  }]  
}
```



# 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

```
locals {
  model = yamldecode(file("./apic.yaml"))
}

module "tenant" {
  source = "../modules/terraform-aci-tenant"

  for_each = toset(
    [for tenant in local.model.apic.tenants : tenant.name]
  )


  model      = local.model
  tenant_name = each.value
}
```

 main.tf

# 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

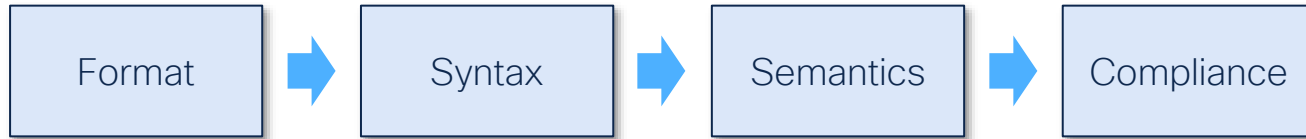
```
apic:  
  tenants:  
    - name: CiscoLive  
      bridge_domains:  
        - name: vlan_101  
        - name: vlan_102  
        - name: vlan_103
```

 tenants.yaml

# 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.



# 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.



# 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

# 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 %}
```



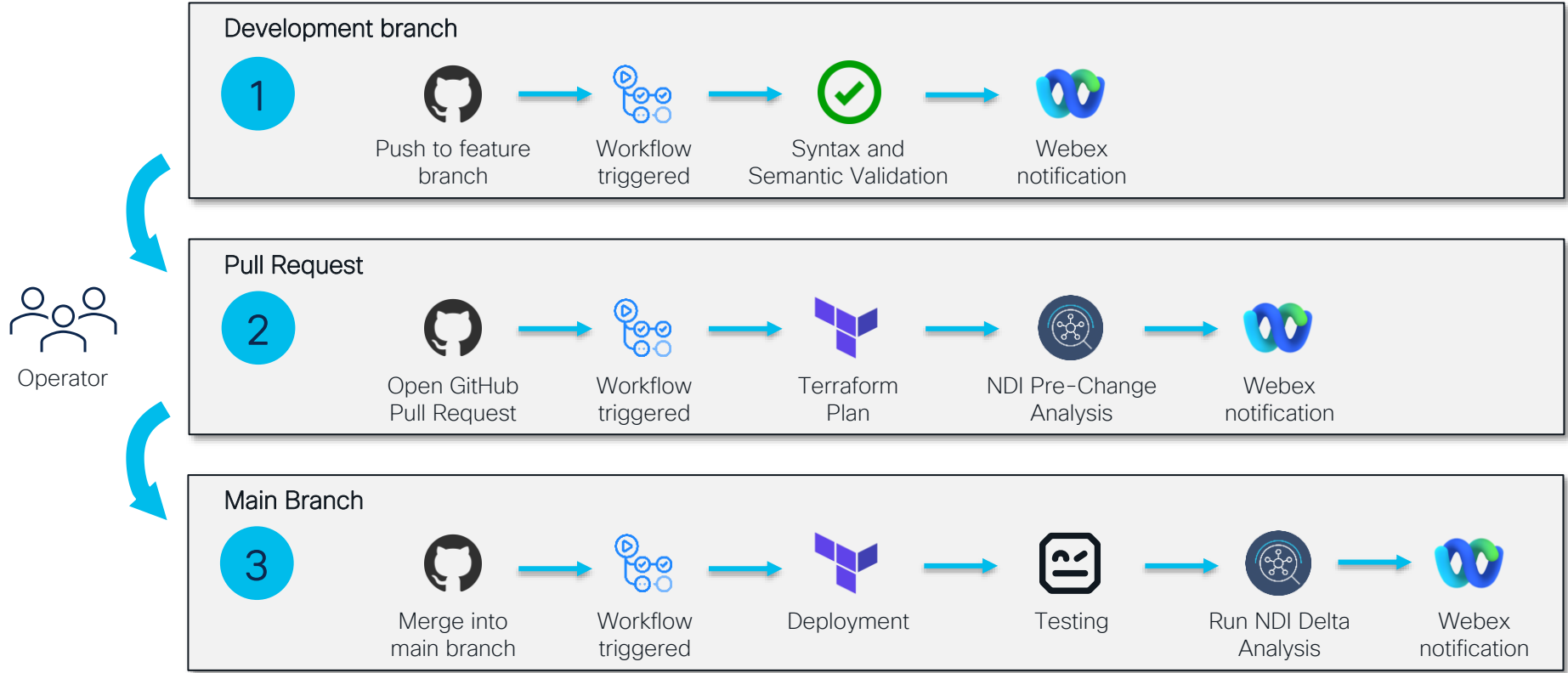
# Idempotency Testing

After applying our changes with **terraform apply** we expect a subsequent **terraform plan** (using the same infrastructure code) to return with no changes. This ensures that all changes have been applied successfully.

This test can be easily integrated into a CI/CD workflow by using the **-detailed-exitcode** flag when executing **terraform plan** which will return a non-zero exit code if changes are detected.

```
$ terraform plan -detailed-exitcode  
  
No changes. Infrastructure is up-to-date.  
  
$
```

# CI/CD Workflow Example



# Demo

<https://github.com/netascode/DEVNET-2097-Demo>



# Lessons Learned

- Think about the data model first, only then start coding
- While doing this use a schema to define it
- Structure and modularize the code
- Separate data from code
- Consider default values and where to define them
- Pre-Change Validation helps to detect mistakes before they have an impact on a production environment
- Automated testing provides proactive feedback
- CI/CD helps standardizing and enforcing a change process

# References

- Nexus-as-Code  
<https://cisco.com/go/nexusascode>
- Demo Repository  
<https://github.com/netascode/DEVNET-2097-Demo>
- ACI Terraform Provider  
<https://registry.terraform.io/providers/CiscoDevNet/aci/latest>
- Pre-Change Validation Tool  
<https://github.com/netascode/iac-validate>
- Test Automation Tool  
<https://github.com/netascode/iac-test>
- NX-OS, IOS-XE, IOS-XR Terraform Providers  
<https://registry.terraform.io/search/providers?q=netascode>

# Complete your Session Survey

- Please complete your session survey after each session. Your feedback is important.
- All surveys can be taken in the Cisco Events Mobile App or by logging in to the Session Catalog and clicking the "Attendee Dashboard" at <https://www.ciscolive.com/emea/learn/sessions/session-catalog.html>



# Continue Your Education



Visit the Cisco Showcase for related demos.



Book your one-on-one Meet the Engineer meeting.



Attend any of the related sessions at the DevNet, Capture the Flag, and Walk-in Labs zones.



Visit the On-Demand Library for more sessions at [ciscolive.com/on-demand](https://ciscolive.com/on-demand).



The bridge to possible

# Thank you

CISCO *Live!*



CISCO *Live!*

ALL IN