ARISTA

WS:C2 Infrahub + AVD Design Driven Automation

ARISTA

What is Arista AVD?

ARISTA

Your Workshop Instructors



Noel Thompson

New Zealand



Kyle Bush

Texas



Marcus McPeek

Texas

ARISTA

2019

The Complexity Crisis

EVPN/VXLAN fabrics were complex to deploy.

Design guides existed, but turning them into configs was manual, error-prone, and slow.

AVD was built to change that

Encode Arista's best practices into a data model — describe ***WHAT*** you want, not ***HOW*** to configure it.

Open Source | Arista Maintained | TAC Supported

Campus | Data Center | WAN | SD-WAN

Configuration Generation
Best practice EOS configs

Design Best Practices
Architectural knowledge baked in

Deploy via API
Via CloudVision or direct

Automated Backups
Device state captured

Auto Documentation
Accurate fabric docs generated every time

Automated Testing
Pre and post-change validation

Digital Twin Generation
Model and test before production

What is Arista AVD?

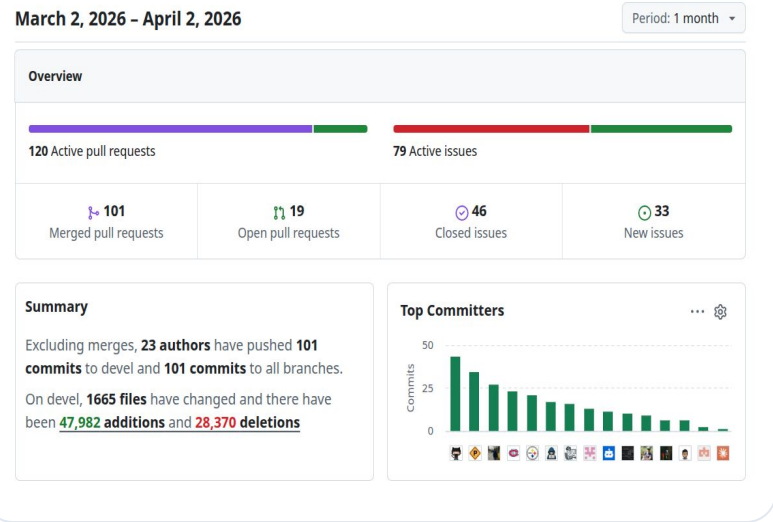
Arista AVD - Architect Validate Deploy

- **Open-source project**
- **Python Network Automation Framework**
  (packaged as a certified Ansible collection)



arista.avd
Ansible Certified
Since November 2020

pyavd
PyPI Published
Since December 2023



144
Releases to date
AVD 6.5.0 release Nov 2026

134
Community Contributors
287 Forks

What is Arista AVD?

AVD IS...

A Data-Driven Framework

Uses structured YAML data to define network intent independently of the device configuration.

A Best-Practice Engine

Encapsulates Arista's validated designs into automated templates for any network design.

Open-Source | Arista Maintained | Community-Led

Publicly available on GitHub, maintained by Arista devs, and driven by the networking community.

Highly Extensible

Compatible with Ansible, Python (PyAVD), and any CI/CD pipelines.

TAC Supported

Covered by A-Care & software life cycle policy.

AVD IS NOT...

A GUI-Based Tool | Appliance

It is a "Configuration as Code" toolset meant for CLI and automation-first workflows.

A Monitoring Platform

While it validates intent, real-time telemetry is handled by CloudVision or other NMS.

Just a Config Generator

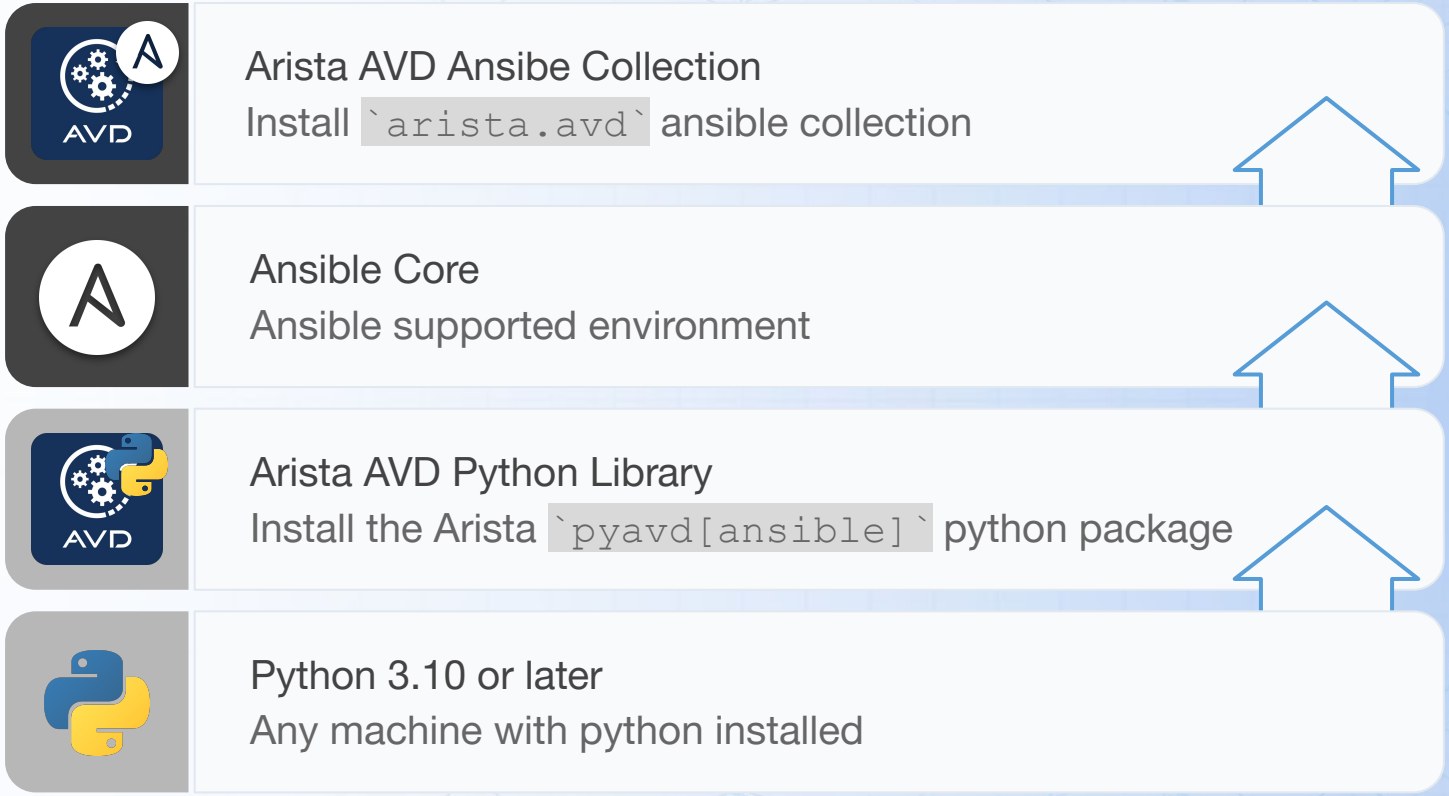
It provides validation, documentation, and digital twin testing—not just raw config files.

Only for large scale networks

It isn't *just* for large scale networks — it's for networks that want to be managed like one.

AVD Installation Components

What's required to run AVD



 Copyright © Arista 2026. All rights reserved.

ARISTA

From Data Model to Running Network



YOU describe your intent *eos_designs* **AVD handles the complexity** *eos_cli_config_gen* **YOU get configs ready to deploy**

Your input (fabric.yml)

```
# Layer 3 Leaf
type: l3leaf

l3leaf:
 defaults:
   # IPs for P2P links (default /31 carve up)
   uplink_ipv4_pool: 10.255.0.0/23
```

AVD output (leaf1.cfg)

```
!
interface Ethernet2
  description P2P_s1-spine1_Ethernet2
  no shutdown
  mtu 1500
  no switchport
  ip address 10.255.0.2/31
!
```

CONFIGURATION ISN'T THE ONLY THING:

Documentation
Fabric docs auto-generated every run

Automated Tests
Pre and post-change validation

Digital Twin
Test your fabric before production

Deployment
Via EOS eAPI or CloudVision

 Copyright © Arista 2026. All rights reserved.

Confidential.

ARISTA

ARISTA

AVD Schema

What is a Schema?

- For a network engineer automating infrastructure, a schema is a strict blueprint and gatekeeper for your configuration variables.

- In Arista AVD, the schema is a set of proprietary rules that validates every input variable like VLAN IDs, BGP AS numbers, or IP subnets before any EOS configuration is generated.

- Instead of discovering a typo or a type error when a switch rejects a command, the schema catches it at the very beginning of the pipeline.

 Copyright © Arista 2026. All rights reserved.

Confidential.

ARISTA

AVD Schema

- **Network Centric**
- Inspired by:
  - JSON Schema
  - Ansible Argument Spec
  - CloudVision Studios
- Provides **unique features** which are not supported or possible with any of the mentioned schema formats
- **Support for:**
  - dynamic keys
  - primary key
  - automatic type conversion

Doc: input variable validation

```yaml
type: dict
$defs:
  node_type_keys:
    type: list
    primary_key: key
    items:
      type: dict
      keys:
        key:
          type: str
        type:
          type: str
          description: Type value matching this node_type_key.
        connected_endpoints:
          type: bool
          default: false
          description: Are endpoints connected to this node type.
        default_evpn_role:
          type: str
          valid_values:
            - none
            - client
            - server
          default: none
          description: Default evpn_role. Can be overridden in topology vars.
        default_ptp_priority1:
          type: int
          min: 0
          max: 255
          default: 127
          description: Default PTP priority 1
        default_underlay_routing_protocol:
          documentation_options:
            table: node-type-keys
          type: str
          default: "ebgp"
          convert_to_lower_case: true
          $ref: "eos_designs#/keys/underlay_routing_protocol"
          description: |
            Set the default underlay routing_protocol.
            Can be overridden by setting "underlay_routing_protocol" host/group_vars.
```

 Copyright © Arista 2026. All rights reserved.

Confidential.

ARISTA

Data Models & Algorithms

Data Models

- **Opinionated**: Reflects our own specific perspectives and rules by use case
- **Abstract**: Strips away unnecessary complexity to focus only on relevant information.
- **Structured**: Clearly defines entities, their attributes, and the strict relationships between them.
- **Purpose-driven:** Designed and optimized to support specific use cases - like network configuration.

 Copyright © Arista 2026. All rights reserved.

Confidential.

ARISTA

What makes a good data model?

 Copyright © Arista 2026. All rights reserved.

Confidential.

Building Data Models

Translating traditional CLI syntax into structured YAML data models

1. CLI CONFIGURATION



CLI Terminal

```
# Configure DNS Server
ip name-server 9.9.9.9
```

COULD BE REPRESENTED AS

2. STRUCTURED DATA MODEL

schema.yml

```
# YAML Data Model Representation
name_server: 9.9.9.9
```

 Copyright © Arista 2026. All rights reserved.

Confidential.

ARISTA

Building Data Models

Translating traditional CLI syntax into structured YAML data models

1. CLI CONFIGURATION



CLI Terminal

```
# Configure DNS Server
ip name-server 9.9.9.9
ip name-server 8.8.8.8
```

COULD BE REPRESENTED AS

2. STRUCTURED DATA MODEL

schema.yml

```
# YAML Data Model Representation
name_servers: [9.9.9.9, 8.8.8.8]
```

 Copyright © Arista 2026. All rights reserved.

Confidential.

ARISTA

Building Data Models

Translating traditional CLI syntax into structured YAML data models

1. CLI CONFIGURATION



CLI Terminal

```
# Configure DNS Server

ip name-server vrf default 9.9.9.9
ip name-server vrf MGMT 8.8.8.8
```

COULD BE REPRESENTED AS

2. STRUCTURED DATA MODEL

ok.yml | better.yml | best.yml

```
# YAML Data Model Representation

name_servers: [default, 9.9.9.9, MGMT,
8.8.8.8]


name_servers:

- default
- 9.9.9.9
- MGMT
- 8.8.8.8
```

 Copyright © Arista 2026. All rights reserved.

Confidential.

ARISTA

Building Data Models

Translating traditional CLI syntax into structured YAML data models

1. CLI CONFIGURATION



CLI Terminal

```
# Configure DNS Server

ip name-server vrf default 9.9.9.9
ip name-server vrf MGMT 8.8.8.8
```

COULD BE REPRESENTED AS

2. STRUCTURED DATA MODEL

ok.yml | better.yml | best.yml

```
# YAML Data Model Representation

name_servers:
  servers: [9.9.9.9,8.8.8.8]
  vrfs: [default,MGMT]

name_servers:
  servers:
    - 9.9.9.9
    - 8.8.8.8
  vrfs:
    - default
    - MGMT
```

 Copyright © Arista 2026. All rights reserved.

Confidential.

ARISTA

Building Data Models

Translating traditional CLI syntax into structured YAML data models

1. CLI CONFIGURATION



CLI Terminal

```
# Configure DNS Server

ip name-server vrf default 9.9.9.9
ip name-server vrf MGMT 8.8.8.8
```

COULD BE REPRESENTED AS

2. STRUCTURED DATA MODEL

ok.yml | better.yml | best.yml

```
# YAML Data Model Representation

name_servers:

  - ipv4: 9.9.9.9
    vrf: default

  - ipv4: 8.8.8.8
    vrf: MGMT
```

 Copyright © Arista 2026. All rights reserved.

Confidential.

ARISTA

Building Data Models

Translating traditional CLI syntax into structured YAML data models

1. CLI CONFIGURATION



CLI Terminal

```
# Configure DNS Server

ip name-server vrf default 9.9.9.9 priority 1
ip name-server vrf MGMT 8.8.8.8
```

COULD BE REPRESENTED AS

2. STRUCTURED DATA MODEL

ok.yml | better.yml | best.yml

```yaml
# YAML Data Model Representation

name_servers:

  - ipv4: 9.9.9.9
    vrf: default
    priority: 1

  - ipv4: 8.8.8.8
    vrf: MGMT
```

 Copyright © Arista 2026. All rights reserved.

Confidential.

ARISTA

Generating CLI Configuration

Combining structured YAML data models with Jinja2 templates to automate EOS switch configs

schema.yml

```yaml
name_servers:
  - ipv4: 9.9.9.9
    vrf: default

  - ipv4: 8.8.8.8
    vrf: MGMT
```

+

cli_template.j2

```
{# Jinja2 Template for DNS Config #}
{% for server in name_servers %}
ip name-server vrf {{ server[vrf] }} {{ server[ipv4] }}
{% endfor %}
```

→ RENDERING ENGINE

1. CLI CONFIGURATION



```
!
ip name-server vrf default 9.9.9.9
ip name-server vrf MGMT 8.8.8.8
!
```

✓ **Valid CLI Configuration Output** Target: eos-sw1

 Copyright © Arista 2026. All rights reserved.

Confidential.

ARISTA

Generating Documentation

Combining structured YAML data models with Jinja2 templates to generate documentation

schema.yml

```yaml
name_servers:
  - ipv4: 9.9.9.9
    vrf: default

  - ipv4: 8.8.8.8
    vrf: MGMT
```

+

markdown_template.j2

```
# Name Servers
My name servers are
| Server | Vrf |
| – | – |
{% for server in name_servers %}
| {{ server[ipv4] }} | {{ server[vrf] }} |
{% endfor %}
```

→ **RENDERING ENGINE**

DOCUMENTATION OUTPUT

Fabric Switches and Management IP

POD	Type	Node	Management IP	Platform	Provisioned in CloudVision	Serial Number
DC_FABRIC	l2leaf	LEAF1	192.168.0.13/24	vEOS	Provisioned	-
DC_FABRIC	l2leaf	LEAF2	192.168.0.14/24	vEOS	Provisioned	-
DC_FABRIC	l2leaf	LEAF3	192.168.0.17/24	vEOS	Provisioned	-
DC_FABRIC	l2leaf	LEAF4	192.168.0.18/24	vEOS	Provisioned	-
DC_FABRIC	l3spine	SPINE1	192.168.0.11/24	vEOS	Provisioned	-
DC_FABRIC	l3spine	SPINE2	192.168.0.12/24	vEOS	Provisioned	-

Provision status is based on Ansible inventory declaration and do not represent real status from CloudVision.

Fabric Switches with inband Management IP

POD	Type	Node	Management IP	Inband Interface
DC_FABRIC	l2leaf	LEAF1	10.255.255.4/24	Vlan255
DC_FABRIC	l2leaf	LEAF2	10.255.255.5/24	Vlan255

✓ **Generated Documentation Output** Format: Markdown

 Copyright © Arista 2026. All rights reserved.

Confidential.

ARISTA

Generating Testing Catalogs

Combining structured YAML data models with Jinja2 templates for ANTA automated test suites

schema.yml

```yaml
name_servers:
  - ipv4: 9.9.9.9
    vrf: default

  - ipv4: 8.8.8.8
    vrf: MGMT
```

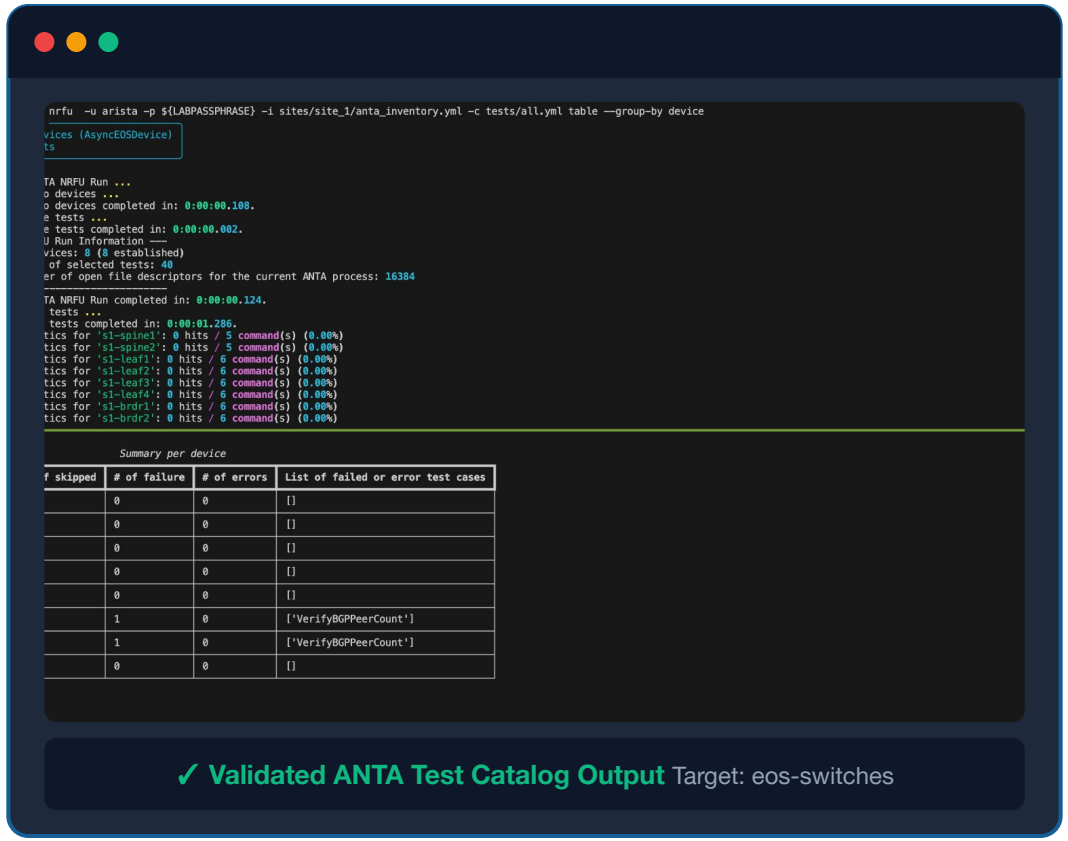
+

testing_template.j2

```
anta.tests.services:
{% if name_servers %}
  - VerifyDNSServers:
      dns_servers:
{%      for server in name_servers %}
        - server_address: {{ server[ipv4] }}
          vrf: {{ server[vrf] }}
          priority: {{ server[priority] }}
{%      endfor %}
{% endif %}
```

→ **RENDERING ENGINE**

AUTOMATED TEST CATALOG & VALIDATION



 Copyright © Arista 2026. All rights reserved.

Confidential.

ARISTA

Network Wide Models

Abstracting network services across an entire fabric

4 **Input Variables**
To generate entire configuration

=

14 **EOS Config Lines**
Generated per switch

=

1,400 **Lines of Config**
For ***100 VTEPs*** (leafs)

tenants.yml

```
---
tenants:
  - name: S1_FABRIC
    mac_vrf_vni_base: 10000
    vrfs:
      - name: OVERLAY
        vrf_vni: 10
        svis:

          - id: 10
            name: 'Ten'
            enabled: true
            ip_address_virtual: 10.10.10.1/24

          - id: 20
            name: 'Twenty'
            enabled: true
            ip_address_virtual: 10.20.20.1/24
```

→

switch.cfg

```
!
vlan 20
   name Twenty
!
interface Vlan10
   description Twenty
   no shutdown
   vrf OVERLAY
   ip address virtual 10.20.20.1/24
!
interface Vxlan1
   vxlan vlan 20 vni 10020
!
router bgp 65101
   vlan 20
      rd 10.255.254.3:10020
      route-target both 10020:10020
      redistribute learned
!
```

→

leaf-001.cfg to 100.cfg

```
vlan 20
  name Twenty
interface Vlan20
  ip address virtual 10.20.20.1/24
router bgp 65101
  vlan 20
    rd 10.255.254.3:10020
    route-target both 10020:10020
    redistribute learned
```

Automated Scale Benefit

Single source of truth automatically renders across all switches.

350x Efficiency Gain

Zero manual CLI copy-paste errors

 Copyright © Arista 2026. All rights reserved.

Confidential.

ARISTA

Network Wide Models

Example using network_ports and port_profiles data models to generate configs for 960 ports

15 **Input Variables**
Port Configuration and range with regex!

+

19 **Input Variable**
Port Profiles

=

16,320 **Lines of Config**
Expands to **960 ports** (4 Chassis with 240 ports)

ports.yml

```
---
network_ports:

  - switches:
      - sw-1-[1234]
    switch_ports:
      - Ethernet3-7/1-48
    profile: PP-DOT1X
    native_vlan: 500
    structured_config:
      phone:
        trunk: untagged
        vlan: 700
    dot1x:
      authentication_failure:
        action: allow
        allow_vlan: 800
```



profiles.yml

```
port_profiles:

  PP-DOT1X:
    mode: "trunk phone"
    spanning_tree_portfast: edge
    spanning_tree_bpduguard: enabled
    dot1x:
      port_control: auto
      reauthentication: true
      pae:
        mode: authenticator
      host_mode:
        mode: multi-host
        multi_host_authenticated: true
      mac_based_authentication:
        enabled: true
      timeout:
        reauth_period: server
        tx_period: 3
      reauthorization_request_limit: 3
```

profiles.yml

```
interface Ethernet3/1
    switchport trunk native vlan 500
    switchport phone vlan 700
    switchport phone trunk untagged
    switchport mode trunk phone
    spanning-tree portfast
    spanning-tree bpduguard enable
    dot1x pae authenticator
    dot1x authentication failure action "..."
    dot1x reauthentication
    dot1x port-control auto
    dot1x host-mode multi-host authenticated
    dot1x mac based authentication
    dot1x timeout tx-period 3
    dot1x timeout reauth-period server
    dot1x reauthorization request limit 3
```

Automated Scale Benefit

480x Efficiency Gain

Zero manual CLI copy-paste errors

 Copyright © Arista 2026. All rights reserved.

Confidential.

ARISTA

ARISTA

Arista Network Test Automation (ANTA) High Level

What ANTA does

An open-source Python framework for testing the operational state of Arista EOS networks

EOS focused	Reads live device state through the EOS eAPI
Concurrent tests	Runs checks across selected devices asynchronously
Extensible	Combines built-in tests with custom checks
Flexible use	Works through a CLI, a Python library or AVD

 Copyright © Arista 2026. All rights reserved.

Confidential.

Where ANTA fits

Network acceptance	Automates Network Ready For Use (NRFU) checks before handover
After a change	Repeats the same checks after upgrades or configuration changes
Daily operations	Provides consistent health checks on demand or through a scheduler
Delivery pipelines	Adds a validation step with structured results for automated workflows

 Copyright © Arista 2026. All rights reserved.

Confidential.

ARISTA

How ANTA works

01

Inventory

The devices to test, with optional tags for selection

02

Test catalog

The checks to run and their expected values

03

Runner

Concurrent tests that query EOS through eAPI

04

Results

An outcome and supporting detail for each test

The inventory defines where to test. The catalog defines what to test.

 Copyright © Arista 2026. All rights reserved.

Confidential.

What ANTA can validate

Network area	Example checks
Routing	BGP sessions, OSPF neighbours, expected routes
Switching and overlays	MLAG health, VLANs, VXLAN state
Connectivity and interfaces	IP reachability, link state, interface errors
System health	CPU, memory, uptime, EOS version
Services and security	NTP synchronisation, logging, access control checks

The selected tests and their inputs determine the validation scope.

 Copyright © Arista 2026. All rights reserved.

Confidential.

ARISTA

ARISTA

Labs