Introduction to Nornir

Pythonic Network Automation

Jason Belk
Senior Technical Advocate, @renobelk

CISCO Live !

Cisco Webex App

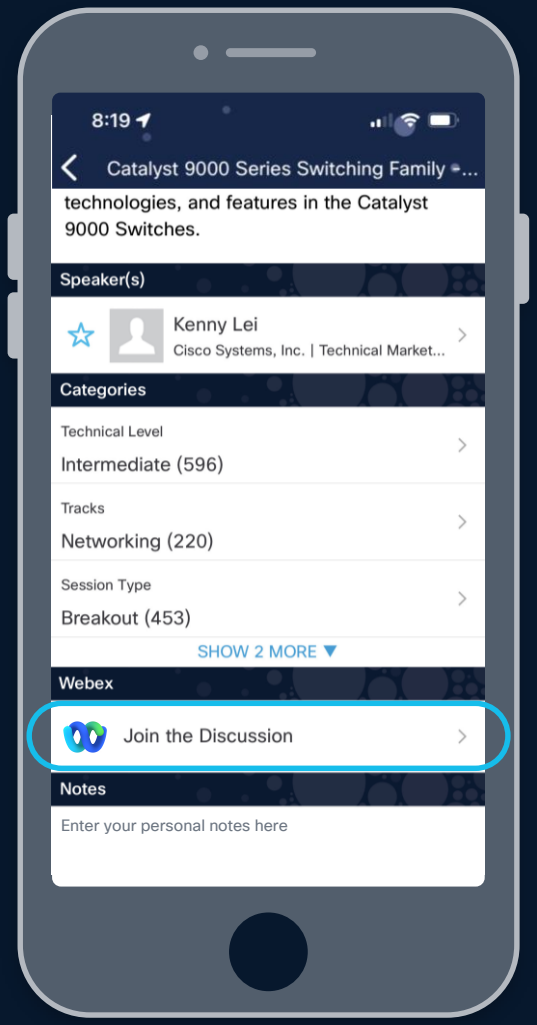
Questions?

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 by the speaker until June 13, 2025.



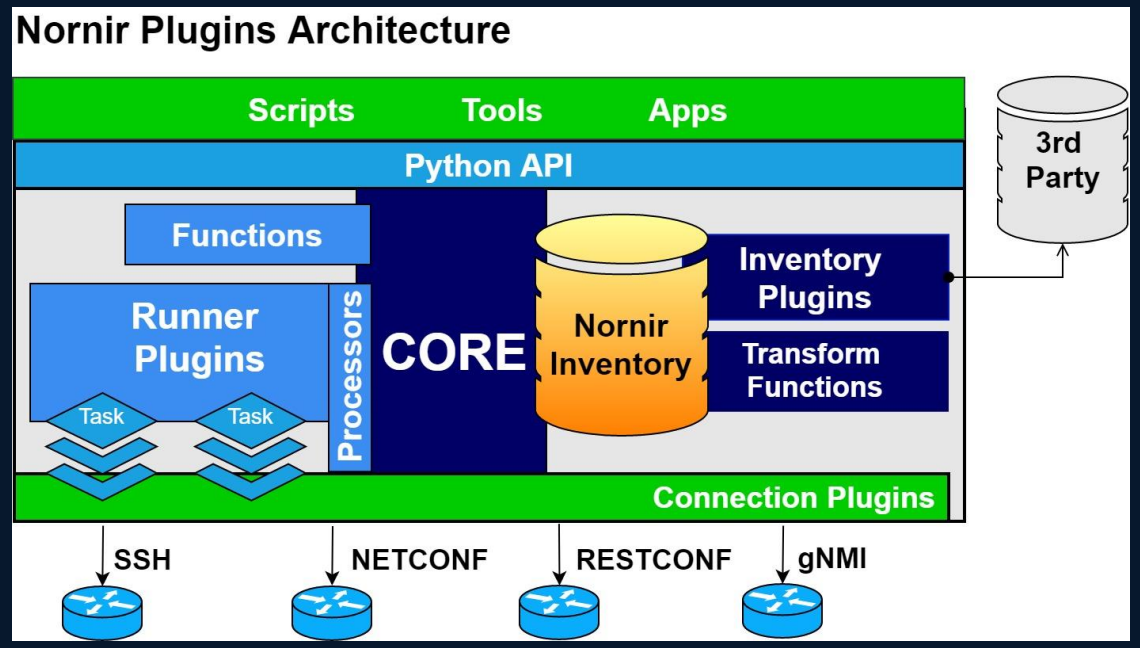
Convenience vs. Control: The Trade-Off Choice

Ansible – Convenient but Limited

Nornir – Pythonic Control

What is Nornir?

Nornir is a pluggable, multi-threaded automation framework with built-in inventory management that helps you operate collections of network devices.



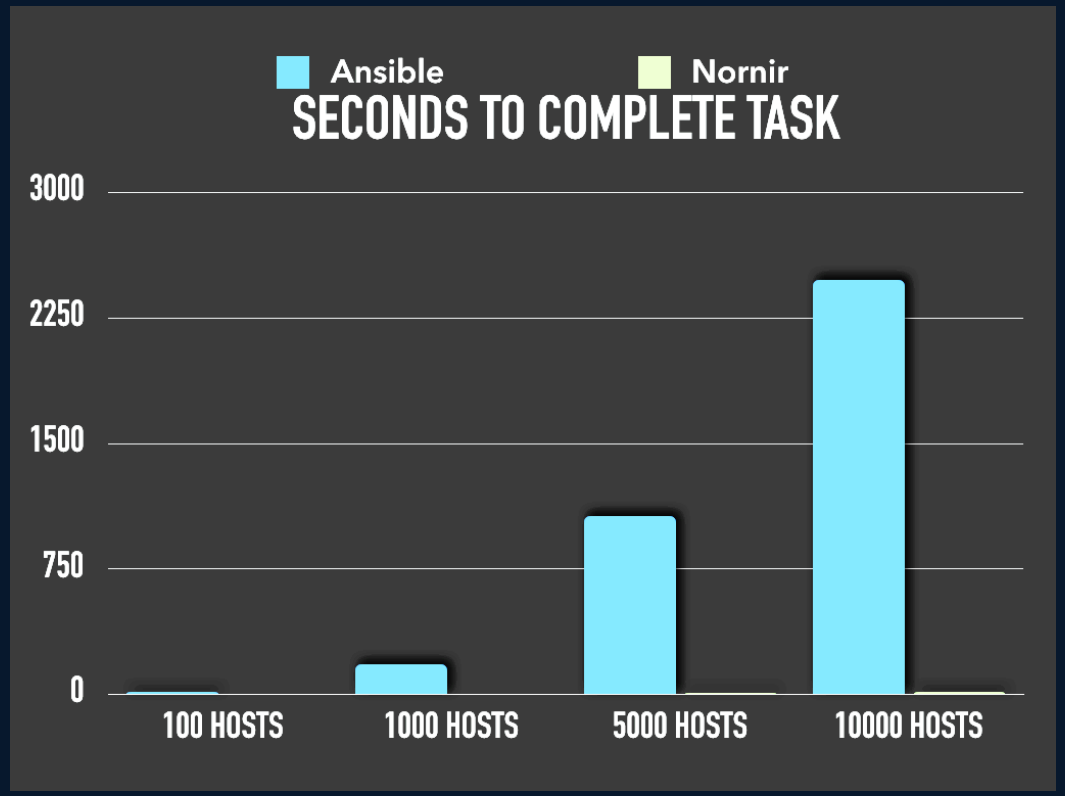
- Nornir Core
  - Inventory (YAML or NetBox)
  - Task Runner (Threads / Orchestration)
  - Plugins (Device Actions)
    - Netmiko
    - Scrapli
    - NAPALM
    - Jinja2

Feature Construct Comparison: Ansible vs Nornir

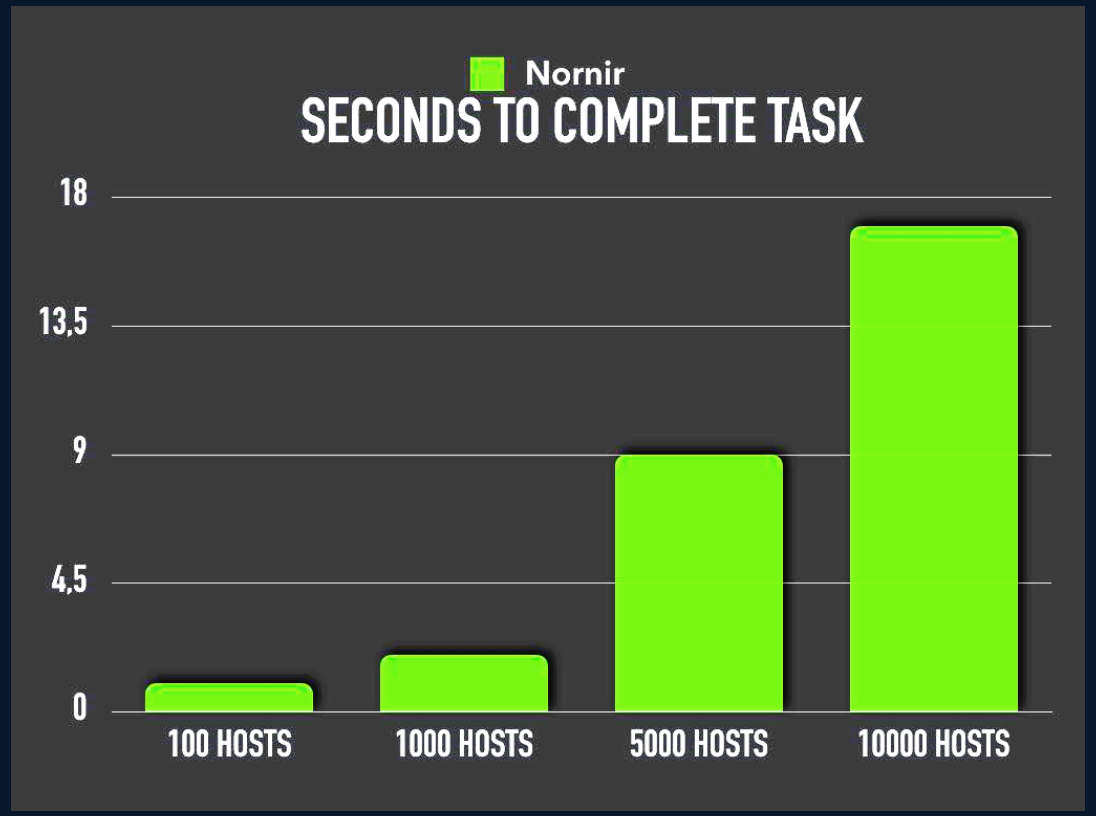
Concept	Ansible	Nornir
Workflow	Playbook (YAML)	Python Script
Task Definition	Module call (e.g. ios_config)	Python function (task.run(...))
Modules / Actions	Ansible Modules	Plugins (Netmiko, Scrapli, NAPALM)
Inventory	INI/YAML/Dynamic	YAML/JSON or Dynamic Plugins
Variables	Host/group vars	Inventory fields, task.host
Conditionals / Logic	when, loop, filters	Full Python logic
Templates	Jinja2	Jinja2 plugin in Python call
Execution Trigger	ansible-playbook pb-ios.yaml	python run_tasks.py

Why Speed Still Matters – Even Below 100 Devices

Performance gap widens dramatically at scale



Nornir scales linearly with predictable performance



"I hit a ceiling with just 90 devices – the issue wasn't scale, it was overhead."

https://networklore.com/ansible-nornir-speed/

Nornir Config (config.yaml)

- Inventory
  - **Plugin**: Specifies the inventory plugin
    - SimpleInventory (YAML files)
    - NetBoxInventory2 (external plugin – NetBox)
    - AnsibleInventory (for reuse from Ansible)
    - Custom (write your own)
  - **Options**:. Optional settings depend on plugin — NetBox plugins may require API keys, filters, etc.
- Runner
  - **Plugin**: Defines how tasks are executed. threaded allows parallel execution.
    - Use threaded for speed, serial for easier debugging
  - **Options**: Sets the number of worker threads.
- Logging (optional)

```yaml
inventory:
  plugin: SimpleInventory
  options:
    host_file: "inventory/hosts.yaml"
    group_file: "inventory/groups.yaml"
    defaults_file: "inventory/defaults.yaml"

runner:
  plugin: threaded
  options:
    num_workers: 10

logging:
  enabled: true
```

Understanding Nornir Inventory

Static Inventory Files (YAML)

```yaml
# defaults.yaml
username: developer
password: C1sco12345
port: 22
```

```yaml
# groups.yaml
ios:
  platform: ios
```

```yaml
# hosts.yaml
ios_device:
  hostname: 10.10.20.48
  groups: [ios]
  data:
    site: "lab1"
    role: "access"
    loopbacks:
      - number: 100
        ip: 10.0.0.1
        mask: 255.255.255.255
```

How Inventory is Merged

```
defaults.yaml
    ↓
groups.yaml (ios)
    ↓
hosts.yaml (ios_device)
```

Accessing Inventory in Python

```python
# Inside a task
task.host.hostname    # "10.10.20.48"
task.host.platform    # "ios"
task.host['username']  # "developer"
```

Inventory Schema

```
{
    "name": "str",
    "connection_options": {
        "$connection_type": {
            "extras": {
                "$key": "$value"},
            "hostname": "str",
            "port": "int",
            "username": "str",
            "password": "str",
            "platform": "str"}},
    "groups": ["$group_name"],
    "data": {
        "$key": "$value"},
    "hostname": "str",
    "port": "int",
    "username": "str",
    "password": "str",
    "platform": "str"
}
```

Running a Task with .run()

- **InitNornir(config_file="...")**
  - Loads config, inventory, and runner setup
- **.run(task=...)**
  - Executes the given function on each host in parallel (threaded by default)
- Nornir passes a task object to your function – includes host data and context
- Use **task.host[...]** to access fields from the merged inventory (hostname, role, etc.)
- The result is returned per host as a structured object – not just printed text

Example: Run simple task across all hosts

```
from nornir import InitNornir

def greet_host(task):
    print(f"Hello from {task.host.name}!")
    print(f"- Platform: {task.host.platform}")
    print(f"- IP Address: {task.host.hostname}")

nr = InitNornir(config_file="config.yaml")
nr.run(task=greet_host)
```

Define a task → Nornir assigns host context → Task runs in thread → Result returned

Unpacking the Nornir Result Object

Result Structure: From .run() to Device Output

```
result = nr.run(task=...)
       ↓
AggregatedResult                ← top-level result (per host)
 └── "ios_device"
      └── MultiResult           ← list of individual task results
           └── Result           ← actual command output, status, etc.
```

Reading Results in Code

```
result["ios_device"]                    # MultiResult
result["ios_device"][0].result          # Output string
result["ios_device"][0].failed          # True/False
result["ios_device"][0].exception       # If an error occurred
```

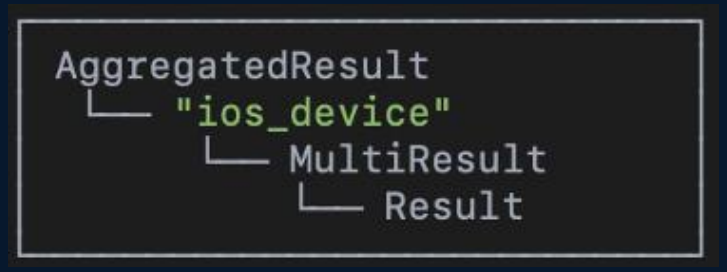
How This Works

- **AggregatedResult** – overall result tree, with one entry per host
- **MultiResult** – list of task results for one host
- **Result** – output from a single task on a single host (string, dict, etc.)
- The result tree mirrors your inventory:
  - result["host"] → [tasks] → result object
- Common attributes:
  - .result → actual output
  - .failed → True/False if task failed
  - .exception → Captures thrown errors

Working with Results in Real Tasks

Example: Inspecting task results per host

```python
for host, multi_result in result.items():
    print(f"\nHost: {host}")
    task_result = multi_result[0]  # Nornir returns a list of task results per host
    print("  .result  →", task_result.result)
    print("  .failed  →", task_result.failed)
    print("  .exception→", task_result.exception)
    if task_result.failed:
        print("This task failed — check .exception for details")
```



How This Works

- Each task returns a **Result** object per host
- You can **access results in code**, not just print them
- Use .result, .failed, and .exception to inspect what happened
- One task, many devices — **each host has its own result**
- This opens the door to logging, logic, and decision-making

Executing a Command with Netmiko + Viewing Results

```python
from nornir import InitNornir
from nornir_utils.plugins.functions import print_result
from nornir_netmiko.tasks import netmiko_send_command

nr = InitNornir(config_file="config.yaml")

result = nr.run(
    task=netmiko_send_command,
    command_string="show version"
)

print_result(result)
```

.run() Basics

- Executes a task on **every host in inventory**
- **kwargs: Passed directly to the task function/plugin
  - In this example: command_string="show version" goes to netmiko_send_command
- Runs in parallel (default: threaded runner)

Netmiko Plugin

- Imported like any other Python module
- Used as a **task**: task=netmiko_send_command
- Handles connection setup, execution, and teardown
- Works with .run() like any custom function

Chaining Tasks with Shared Data

One task calls multiple subtasks in order

```python
def config_chain(task):
    task.run(task=show_hostname)
    task.run(task=set_loopback_description)
```

Extracts hostname and stores it on the host object

```python
def show_hostname(task):
    result = task.run(
        task=netmiko_send_command,
        command_string="show run | include hostname"
    )
    hostname = result[0].result.strip().split()[-1]
    task.host["hostname_from_device"] = hostname
```

Reads hostname and pushes config to the device

```python
def set_loopback_description(task):
    hostname = task.host.get("hostname_from_device", "unknown")
    return task.run(
        task=netmiko_send_config,
        config_commands=[
            "interface Loopback100",
            f"description Host: {hostname}"
        ]
    )
```

How This Works

- Run multiple tasks in sequence using task.run() inside a single function
- Store data between tasks using task.host
- Drive logic based on live device state or previous results
- Each host executes its own task chain independently
- A flexible, Pythonic alternative to static playbook lists

Use Cases

- Gather multiple facts or CLI outputs in a structured format
- Build custom “playbook-like” sequences inside Python
- Pre-checks → config render → post-checks (in one task)
- Build dynamic task chains that adapt per device

Targeting Devices with filter()

Filtering to IOS devices

```
nr = InitNornir(config_file="config.yaml")
nr = nr.filter(platform="ios")
result = nr.run(task=config_chain)
print_result(result)
```

Common Use Cases

- nr.filter(platform="ios")
- nr.filter(role="access")
- nr.filter(groups__contains="lab")
- nr.filter(platform="ios", site="lab1")

Why Use .filter()

- Targets only matching hosts from inventory
- Helps avoid running tasks on incompatible devices
- Works with any host attribute (platform, site, role, group)

Rendering Configs with Jinja2 Templates

Host inventory with custom data fields

```yaml
# hosts.yaml
ios_device:
  hostname: 10.10.20.48
  groups:
    - ios
  data:
    loopbacks:
      - number: 100
        ip: 10.0.0.1
        mask: 255.255.255.255
```

Task that renders config and applies it

```python
def render_and_apply_config(task):
    result = task.run(
        task=template_file,
        template="loopbacks.j2",
        path="templates",
        loopbacks=task.host["loopbacks"]
    )
    config_lines = result.result.splitlines()
    task.run(
        task=netmiko_send_config,
        config_commands=config_lines
    )
```

Loopback Jinja2 template to render

```
{% for loopback in loopbacks %}
interface Loopback{{ loopback.number }}
 ip address {{ loopback.ip }} {{ loopback.mask }}
{% endfor %}
```

How This Works

- template_file renders loopbacks.j2 with host-specific variables
- loopbacks= passes list from task.host["loopbacks"]
- result.result returns the full rendered config as a string
- .splitlines() converts it into a CLI command list
- Data can come from inventory, JSON, YAML, or other sources

Collecting Device Facts with NAPALM + Exporting to CSV

Run NAPALM to pull structured device info

```python
from nornir_napalm.plugins.tasks import napalm_get

result = nr.run(
    task=napalm_get,
    getters=["facts"]
)
```

Write Facts to CSV with Python

```python
with open("device_facts.csv", "w", newline="") as csvfile:
    fieldnames = [
        "host", "hostname", "model", "os_version", "serial_number", "uptime"
    ]
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
    writer.writeheader()
    for host, multi_result in result.items():
        facts = multi_result[0].result.get("facts", {})
        writer.writerow({
            "host": host,
            "hostname": facts.get("hostname", ""),
            "model": facts.get("model", ""),
            "os_version": facts.get("os_version", ""),
            "serial_number": facts.get("serial_number", ""),
            "uptime": facts.get("uptime", "")
        })
```

How This Works

- napalm_get is a Nornir task that returns structured device data
  - In this case, we're using the "facts" getter for basic info like hostname, model, and uptime
- Results are stored in a predictable structure
  - result[host][0].result["facts"] gives you a dictionary of device facts
- Each host's result is a list (MultiResult) even if only one task was run
  - We access the first task result with [0]
- Python makes it easy to export the data
  - Here we use csv.DictWriter() to write device facts to a CSV file

Handling Failures with Real APIs

Example: Querying MTG Card Info from Scryfall

```python
def fetch_card_info(task):
    card_name = task.host.get("mtg_card", "Black Lotus")
    try:
        response = requests.get("https://api.scryfall.com/cards/named", params={"exact": card_name})
        response.raise_for_status()
        return Result(result=f"Found card: {response.json()['flavor_text']}")
    except requests.RequestException as e:
        return Result(result=f"API request failed for card: {card_name}", failed=True, exception=e)
```

How This Works

- Runs a REST API query per host using the requests library
- Card name is pulled from host variable: task.host["mtg_card"]
- If the request fails (404, timeout, etc), it sets .failed = True and captures the exception
- Success returns the flavor text of the card
- Wrap any logic in a task – REST APIs, file ops, tests, and more

Check Out Free Cisco U Tutorials

TUTORIAL

An Introduction to Nornir Automation Framework

By Jason Belk

Free

Beginner 16m

Updated Jan, 2025

TUTORIAL

Enabling Decision-Making in Network Automation Scripts

By Robert Whitaker

Free

Intermediate 1h 5m

Updated Jan, 2025

TUTORIAL

Construct an EEM Applet for Enterprise Automation

By Quinn Snyder

Free

Beginner 48m

Updated Nov, 2024

TUTORIAL

Working with Data Structures in Network Automation Scripts

By Robert Whitaker

Free

Beginner 1h 10m

Updated Jan, 2025

TUTORIAL

Test-Driven Autotmation with Cisco pyATS Using SSH

By John Capobianco

Free

Intermediate 25m

Updated Jan, 2025

TUTORIAL

Integrating a Github Repository with a Jenkins Job

By Mohit Hooda

Free

Intermediate 32m

Updated Jan, 2025

TUTORIAL

Jinja2 Tips & Tricks with Ansible

By Jason Belk

Free

Beginner 10m

Updated Jan, 2025

TUTORIAL

An Introduction to Ansible Roles

By Jason Belk

Free

Beginner 8m Updated Jan, 2025

TUTORIAL

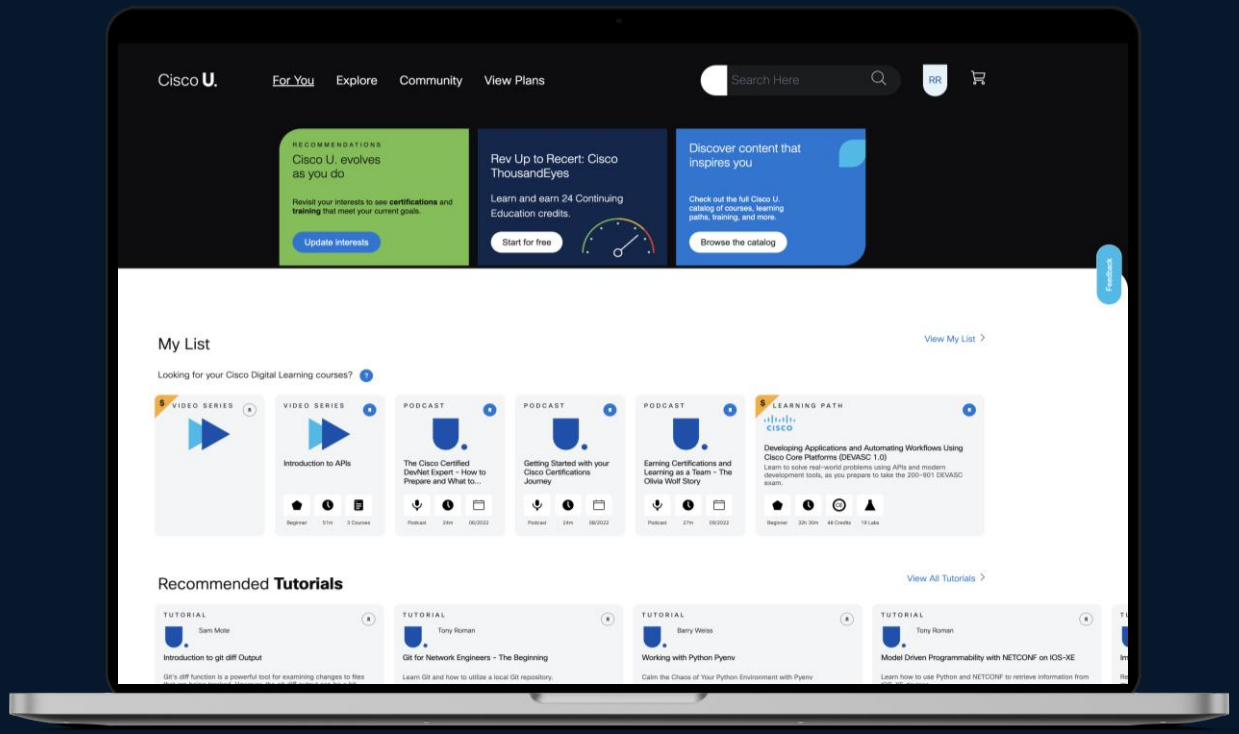
Data Encoding Formats: XML vs JSON vs YAML

By Jason Belk

Free

Beginner 5m Updated Jan, 2025

Explore tutorials, courses at u.cisco.com



Covering AI, Automation, Networking, Security…

From Playbooks to Python: Building Your Workbench

- Nornir is your workbench
  - *Choose the tools, build the workflow, scale when ready*
- You're not locked into modules or syntax — it's just Python
  - *Design automation that fits how your team works*
- Start with one task, grow into a toolkit

What You Can Do Now

- **Connect to devices at scale**
  - Use tasks like netmiko_send_command to audit configs or run diagnostics
- **Generate and deploy configurations**
  - Use Jinja2 templates to create and push changes based on host data
- **Build internal tools on top of Nornir**
  - Create CLI utilities or scripts your team can run to troubleshoot, validate, or deploy

Code From Today: https://github.com/jabelk/nornir-demo

Learn with Cisco AI Skills Survey

CISCO Live !

Complete your session evaluations

Complete a minimum of 4 session surveys and the Overall Event Survey to be entered in a drawing to win 1 of 5 full conference passes to Cisco Live 2026.

Earn 100 points per survey completed and compete on the Cisco Live Challenge leaderboard.

Level up and earn exclusive prizes!

Complete your surveys in the Cisco Live mobile app.

Continue your education

Visit the Cisco Showcase for related demos

Book your one-on-one Meet the Engineer meeting

Attend the interactive education with DevNet, Capture the Flag, and Walk-in Labs

Visit the On-Demand Library for more sessions at www.CiscoLive.com/on-demand

Contact me at: Webex room

Thank you


CISCO