Tackling Config Drift and Compliance Using Automation

Matt Vitale · Managing Consultant, Network to Code

INNUG · September 17, 2026

What We'll Cover

1 The problem

What config drift is and why it’s inevitable

2 How we handle it today

CLI checks, manual audits and why they stall

3 The building blocks

Leveraging automation tools for config compliance

4 Demo

Example of config compliance tools

5 Where AI fits

A first-draft accelerator

6 Where to start

Three things you can start doing right away

Does your network actually look the way you think it does?

A Story We All Know

Change requested

Documented, reviewed, approved

Change implemented

Mostly as approved.
May incur changes, intended or human-error

Time passes

Other changes.
Other engineers.
One-off events.

Reality diverges

The device no longer matches the intent

Nobody did anything wrong. There was no bad actor and no skipped process.

So how would you even know it happened?

Why Config Drift Happens

Emergency changes

Made at 2am under pressure, never documented afterward

“Temporary” configs

Added for a test or a workaround, never removed

Interpretation

Three engineers, one standard, three slightly different results

Silent defaults

An OS upgrade quietly changes a default you relied on

Post-incident fixes

Whatever stopped the bleeding became permanent

Pre-standard gear

Deployed before the current standard existed

Greenfield vs. brownfield

So What Is Your Source of Truth?

WHAT WE SAY

We have documented standards

Every change goes through review

Our diagrams are current

The senior team knows how it should look

WHAT IT ACTUALLY IS

A document last edited three years ago

A ticket describing intent, not outcome

A diagram of the network we planned

Whatever is on the device right now

None of these is a source of truth.

They are all after-the-fact records. A source of truth is where you define intent before it reaches a device.

Why Source of Truth is Important for Automation

- Automation is only as accurate as the data behind it
- A SoT provides one trusted place for inventory and configuration data
- Data can be validated, normalized, and enriched before automation uses it
    - Good data makes good automation
- Connect multiple Systems of Record for a unified inventory view
- Intended state gives operations a reference model to validate against

PART TWO

How we handle this today

The Manual Audit

THE PROCESS

Someone exports running configs

Someone eyeballs them against a standard

Findings go in a spreadsheet

Remediation becomes a ticket queue

Intend to repeat on a regular basis

THE REALITY

It takes days and it is nobody's actual job

Two reviewers reach two different answers

The spreadsheet is stale the day it is finished

The queue never empties

It slips the first quarter as things get busy

Why Manual Doesn't Scale

It's slow

By the time the audit is finished, the network has changed.

Point-in-time findings against a moving target.

It's inconsistent

The standard lives in people's heads, so it varies by reviewer.

Nothing is written down precisely enough to be repeatable.

It's reactive

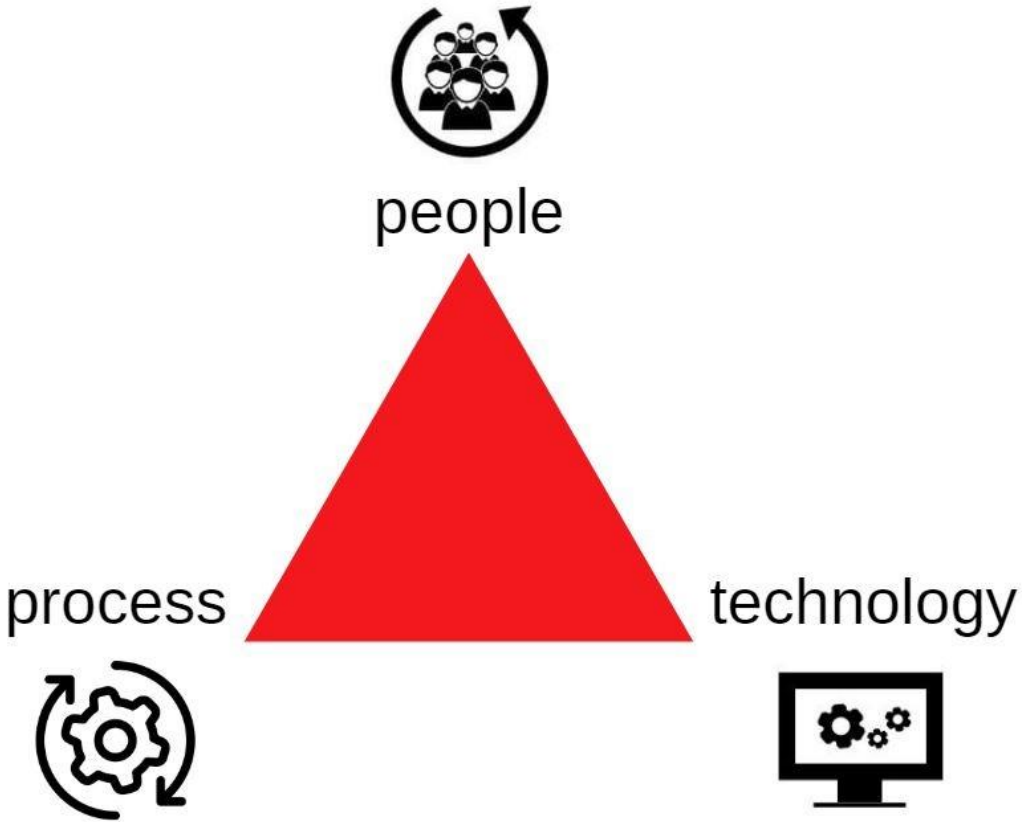
You usually discover the compliance gap during the outage.

Detection happens after the damage, not before.

Scale is not really about device count.

It is about how often the network changes versus how often you can afford to look.

The Big 3



This is a process problem,

not a people or technology problem.

Which is good news, as processes can be automated.

PART THREE

The building blocks

Three concepts. Every tool you will ever evaluate implements these three.

Three Concepts, In Order

1

DEFINE

What should be true?

Your standard, written down as data and logic in a standardized format.

2

COMPARE

What is actually true?

Collect the real config, structure it, diff it against the standard.

3

ACT

What do we do about it?

Report, alert, generate a fix, or remediate. Start small and low impact.

Skip step 1 and you have not built compliance…you have built a very expensive opinion.

Step 1 — Define What “Correct” Looks Like

A golden config is not a file

A saved copy of one good device is a snapshot, not a standard

The standard is the pattern: which values are fixed, which vary by device, and which are simply forbidden

Written as data plus logic, it becomes something a machine can check

It lives in version control, so the standard itself has history and review

THE SHAPE OF A STANDARD

```
NTP
  servers       : exactly 2, from approved list
  authentication: required

SNMP
  version       : v3 only
  community     : must not exist

BGP
  peer password: required on all eBGP
  timers        : identical across peers

# pseudocode, not a product's syntax
```

Intent Lives in a File, Not in Someone's Head

1 — YOUR DATA (what varies per device)

```
site: indy-dc1
role: distribution
snmp:
    version: v3
ntp:
    servers:
        - 8.8.8.8
        - 8.8.4.4
```

2 — YOUR TEMPLATE (what stays fixed)

```
{% for s in ntp_servers %}
ntp server {{ s }}
{% endfor %}
snmp version {{ snmp.version }}
```

3 — RENDERED INTENT (what should be on the box)

```
ntp server 8.8.8.8
ntp server 8.8.4.4
snmp version v3
```

Why this ordering matters

Data and logic stay separate, so one template serves every device.

Change the standard once; every device inherits it.

Rendered intent is now comparable to reality.

Step 2 — Compare Intent to Reality

1. **Collect**
   Pull the running config from the device
2. **Structure**
   Parse text into data you can reason about
3. **Compare**
   Diff actual against rendered intent
4. **Report**
   Per-device, per-rule pass or fail

The step everyone underestimates: structure

Comparing raw text gives you noise — line order, whitespace and defaults all differ harmlessly.
Turning config into structured data first is what makes the comparison meaningful.
This is where most home-grown efforts stall.

The Concepts Don't Change

Write it yourself

A script, a parser, and a comparison you own end to end.

Cheapest to start, most glue to maintain.

Best for learning what the problem actually is.

Open source frameworks

Purpose-built projects handle collection, parsing and reporting for you.

Several mature options exist across the community.

You still supply the standard and the data.

Commercial platforms

Vendor and third-party products bundle this into a supported workflow.

Fastest to a dashboard, least visibility into the mechanics.

Evaluate against the three concepts, not the feature list.

Same three concepts in all three columns.

The only real difference is how much of the plumbing you write and maintain yourself.

Step 3 — Detection Alone Isn't Compliance

Report
A document or dashboard someone has to go read

→

Alert
Drift automatically opens a ticket and notifies

→

Assisted fix
The system generates the remediation, a human approves

→

Closed loop
Automatic remediation inside defined guardrails

increasing automation, increasing trust required

Most teams should aim for level two, and that is a win

Closed-loop remediation is not the goal — a network you can make truthful claims about is the goal.
Every level to the right of 'report' means drift gets found by a machine instead of an outage.

Putting It Together: The Compliance Loop

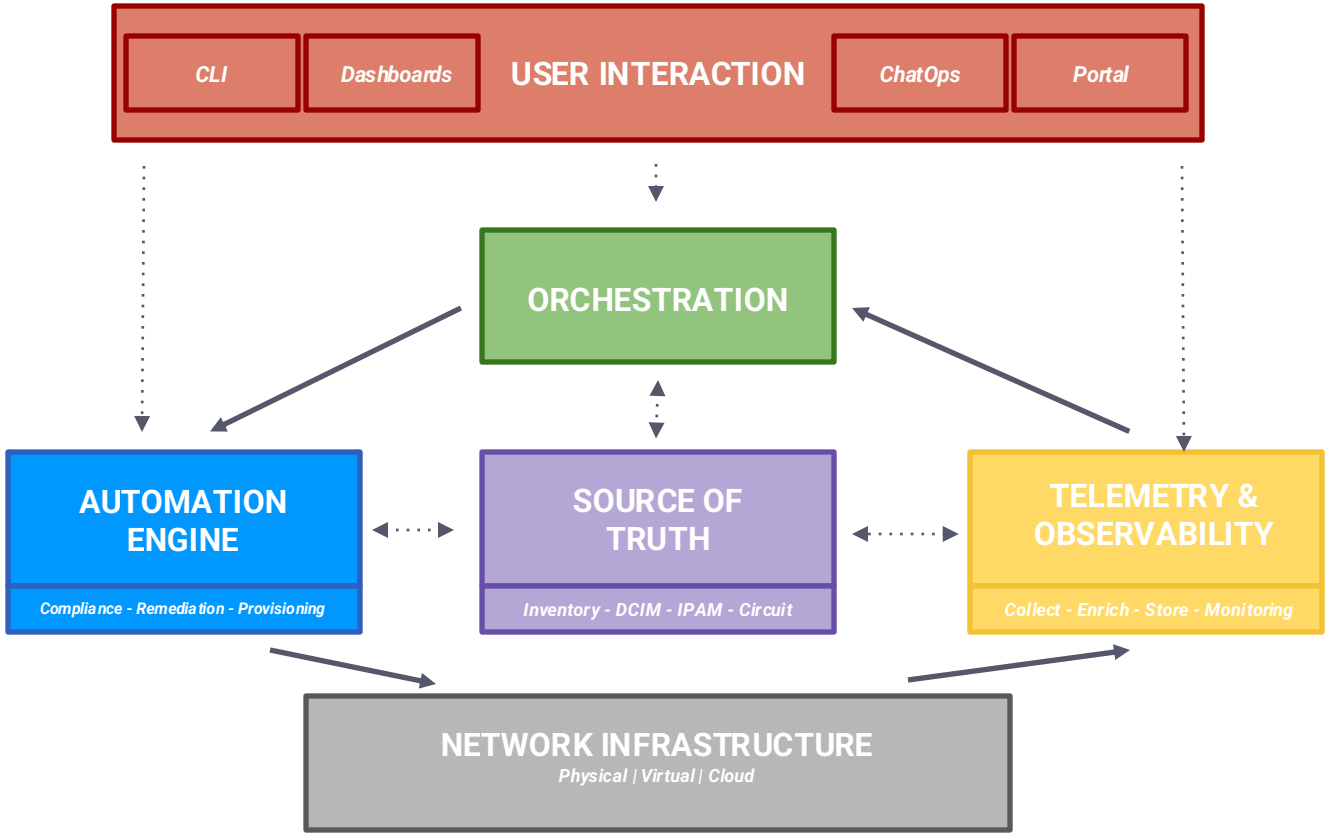
and it runs again — remediation updates the standard, not just the device

The source of truth sits in the middle of all five stages

Intent is defined there, config is generated from it, and drift is measured against it.

Fix a device without fixing the standard and you will drift again next quarter — that feedback arrow is the whole difference between an audit and a practice.

Architecture



PART FOUR

Demo

The Demo in Three Parts

1 Two configs

Same role, same standard, deliberately different in ways you have all seen in production.

2 Three rules

NTP server count, SNMP version, and BGP peer authentication written as structured data.

3 One report

Per-device, per-rule pass or fail, with the offending line called out.

What to notice

Everything a compliance platform sells you is this, plus scale, plus a UI, plus the plumbing you did not want to maintain.

Where AI Actually Helps

It is a first-draft accelerator

Turning an existing config into a starting-point standard you then correct

Drafting the template and the parsing logic so you are editing instead of staring at a blank file

Explaining an unfamiliar config block in plain language

Summarising what changed in a diff before a change window

A realistic ask

“Here is a running config from a distribution switch. Draft a set of compliance rules covering NTP, SNMP and BGP authentication. Flag anything you are not sure about.”

You get a usable draft in seconds. You then spend twenty minutes correcting it, which is far less than the two hours it would have taken to write from nothing.

That is the entire value proposition. Nothing more exciting than that, and nothing less useful.

What AI Won't Do For You

It doesn't know your network

Your addressing, naming, roles and risk tolerance are invisible to it.

It produces plausible config, not correct-for-you config.

This is exactly why the source of truth still has to be yours.

It will invent syntax confidently

Commands that do not exist, parser paths that are wrong, arguments that were never valid.

The output is fluent, which makes the errors harder to spot, not easier.

Anything it writes still has to be tested like code from a stranger.

It can't decide what's correct

It can tell you what a config does and what changed.

It cannot tell you whether that is right for your environment.

That judgement is the job, and it is not going anywhere.

AI changes how fast you write the rules.

It does not change the fact that you need rules, a source of truth, and a review gate.

Where to Get Started

1 **Write down one standard**

Pick your most universal and least impactful setting. NTP, SNMP, syslog, whatever it is. Skip anything that can lock you out. Oh, and skip banner.

Get it out of someone's head and into a file. One standard, not a framework.

2 **Write one check**

A script or tool that reads a config and compares, showing pass or fail against that one standard.

It can be ugly. The goal is to make the concept concrete, not to build a platform.

3 **Make it run without you**

A scheduled job, a pre-change step, anything that does not depend on a human remembering.

Once it runs on a schedule, iterate on it further.

Nobody's first compliance check covered their whole network. Nor will their second, or third, or tenth.

Tools


nautobot


python


CISCO


OpsMill


ANSIBLE


CloudVision
ARISTA


netbox


git

Key Takeaways

1 Drift is the default

It is not a sign of a sloppy team. It is what happens to any network nobody is actively checking.

2 Define before you detect

You cannot compare against a standard that only exists in conversation.

3 Structure beats string diffs

Parsing config into data is the step that makes comparison meaningful.

4 A loop, not an audit

The value is in it running on a schedule without you.

Questions?

Matt Vitale
Managing Consultant · Network to Code
matt.vitale@networktocode.com