27.11.2025

# PARIS

FRANCE-IX TOUR

15 ANS

ÉDITION SPÉCIALE

# Building trustworthy network automation, from principles to practice

Damien Garros - Co-founder & CEO

OPSMILL

# About me : Damien Garros

Co-Founder and CEO of OpsMill

Creator of Infrahub, a next generation Infrastructure data management platform (Source of Truth)

Focused on Infrastructure as Code, Automation & Observability for 12+ years

Previously leading Technical Architecture at Network to Code

OpsMill
Simplifying Complexity, Enabling Agility

INFRAHUB

damiengarros

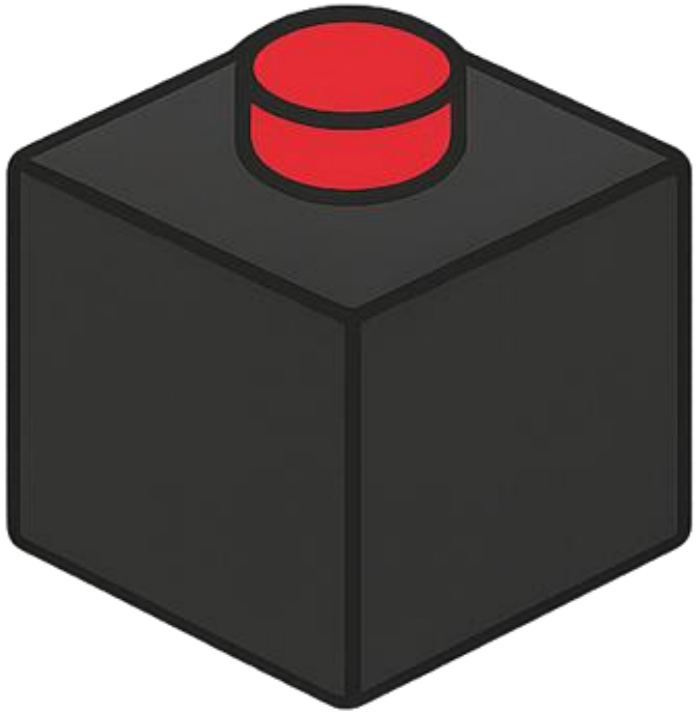
@dgarros

# Goals of this presentation

Provide an overview and the fundamental knowledge for both Automation Users and Automation Builders to collaborate more efficiently and ultimately build better automation systems.

# Trust
# is essential for successful network automation adoption.

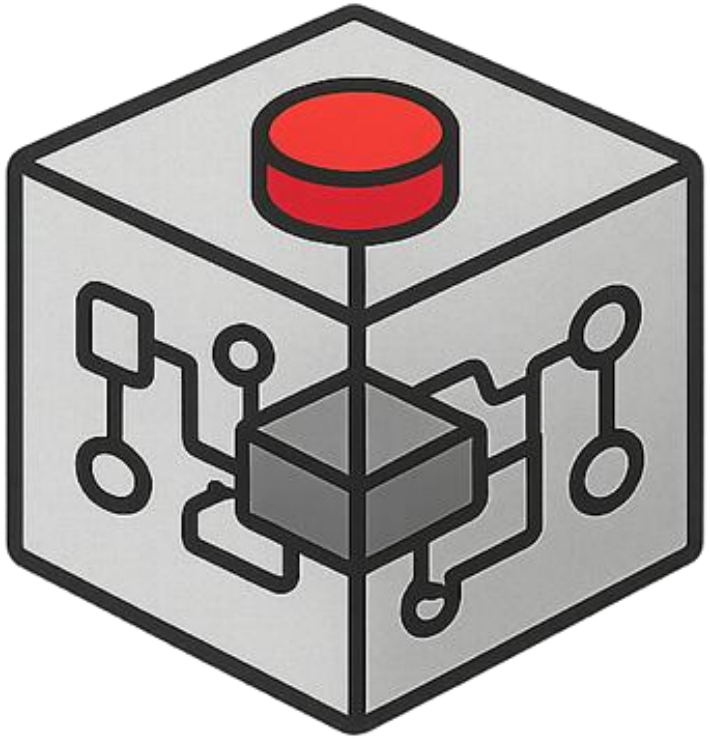
# Press the button to upgrade your network



Software Upgrade

Automation User

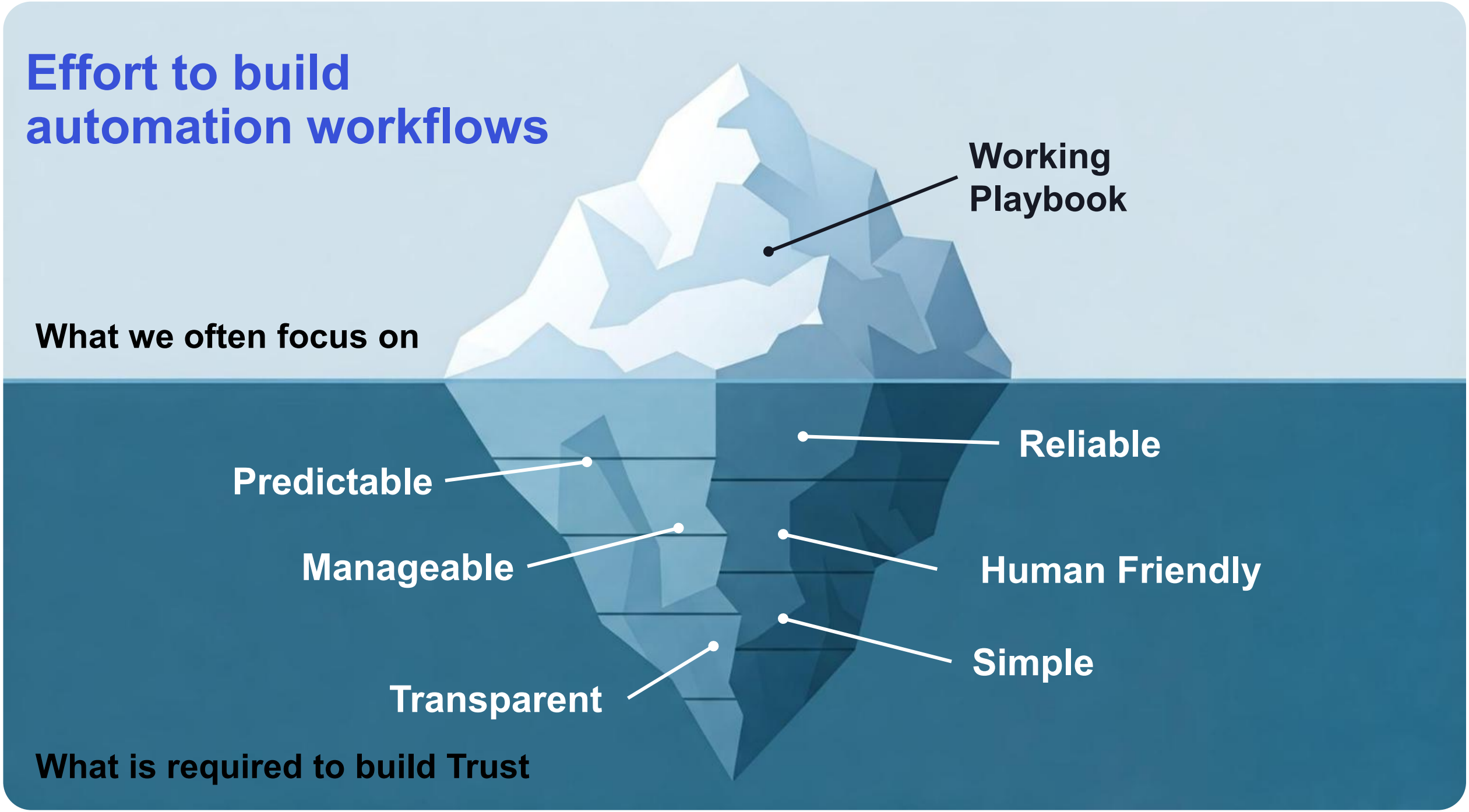
# The person who developed it probably has a completely different perspective on it



**Software Upgrade**

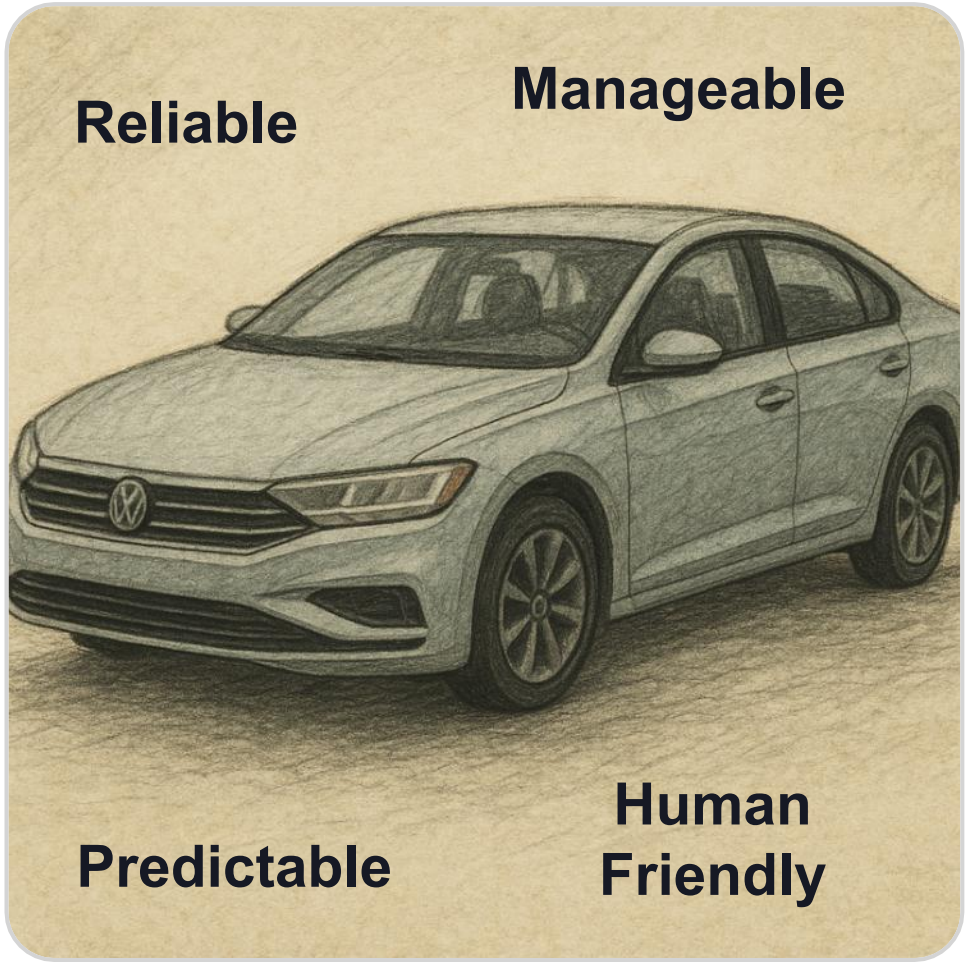
**Automation Developer**

# Effort to build automation workflows



# Which cars do you trust the most ?

## Another perspective on this topic



# Main principles to build trust

## Predictable

Automation should produce consistent and repeatable outcomes every time it runs.

## Manageable

Systems and workflows should be easy to configure, control, and update without hidden complexity.

## Transparent

Automation should clearly show what it will do and what it has done — no surprises.

## Simple

Solutions should avoid unnecessary complexity, making them easier to understand, audit, and maintain.

## Reliable

Automation must handle failures gracefully and ensure that critical operations complete successfully.

## Human Friendly

Interfaces and experiences should be designed with people in mind — intuitive, safe, and supportive of decision-making.

**Trust comes from visibility, control, and graceful failure handling — not just from correct execution.**

# Built on mistakes. Refined by experience.

This presentation present some hard-earned knowledge based on years of trying and making mistakes.

Building automation that’s predictable, manageable, transparent, and reliable isn’t easy.

It takes time, and it takes care — but every step forward matters.

# Design Principles of Trustworthy Automation

# Idempotency

Definition

**Running the same operation multiple times has the same effect as running it once.**

Idempotency is one of the cornerstone of reliability and simplicity in automation systems.

# Example of Idempotency in networking

**NOT idempotent**

I need an IP address ->
<- 10.0.0.1

I need an IP address ->
<- 10.0.0.2

I need an IP address ->
<- 10.0.0.3

---

**Idempotent**

I need an IP address ->
<- 10.0.0.1

I need an IP address ->
<- 10.0.0.1

I need an IP address ->
<- 10.0.0.1

# Example of Idempotency in networking

Idempotency uses a declarative approach to **move the complexity of managing the state from the client .. to the server**

The laptop doesn’t need to know the current state of the system.
The complexity is managed within the server to understand what needs to be done.

My name is Bob and I need an IP address ->
<- 10.0.0.1

My name is Bob and I need an IP address ->
<- 10.0.0.1

**Bob = 10.0.0.1**

# Dry Runs

Definition

**Show users exactly what will change before anything is executed**

Builds confidence and reduces fear of unintended consequences.

# Dry Run mode (AKA check mode)

Before executing any changes, the automation shows exactly what it would do, without actually doing it.

This gives the operator a chance to review, approve, and catch mistakes early.

**“Here’s the diff - do you want to proceed?”**

# Dry Run mode - examples


Ansible includes 2 options
--diff & --check

Check each modules for support

```
@@ -7,7 +7,7 @@

 access-list 101 permit tcp any host 192.168.1.1 eq 80
 access-list 101 permit tcp any host 192.168.1.1 eq 443
-access-list 101 permit ip any any
+access-list 101 deny ip any any
 access-list 101 remark End of ACL
```


Terraform plan, a built-in feature that is supported on all providers

```
# aws_instance.example will be created

+ ami           = "ami-abc123"
+ instance_type = "t3.micro"
```


kubectl diff or ArgoCD show diffs between current cluster state and the desired YAML.

```
spec:
  replicas: 2 -> 3
```

# Transactional

Definition

**Group changes so they either all succeed or can be rolled back cleanly if something fails.**

Prevents partial or broken changes

# Transactional

Transactional automation means grouping a set of changes so they either:

**All succeed** (commit) → and the system moves to the new desired state

**Or none are applied** (rollback) → leaving the system unchanged if something fails

If failure occurs partway through,
the automation ensures no “half-applied” or “broken” states remain.

Rollback capabilities extend this by allowing the system to revert changes after they have been committed if issues are detected later.

# Design Principles to build Trust

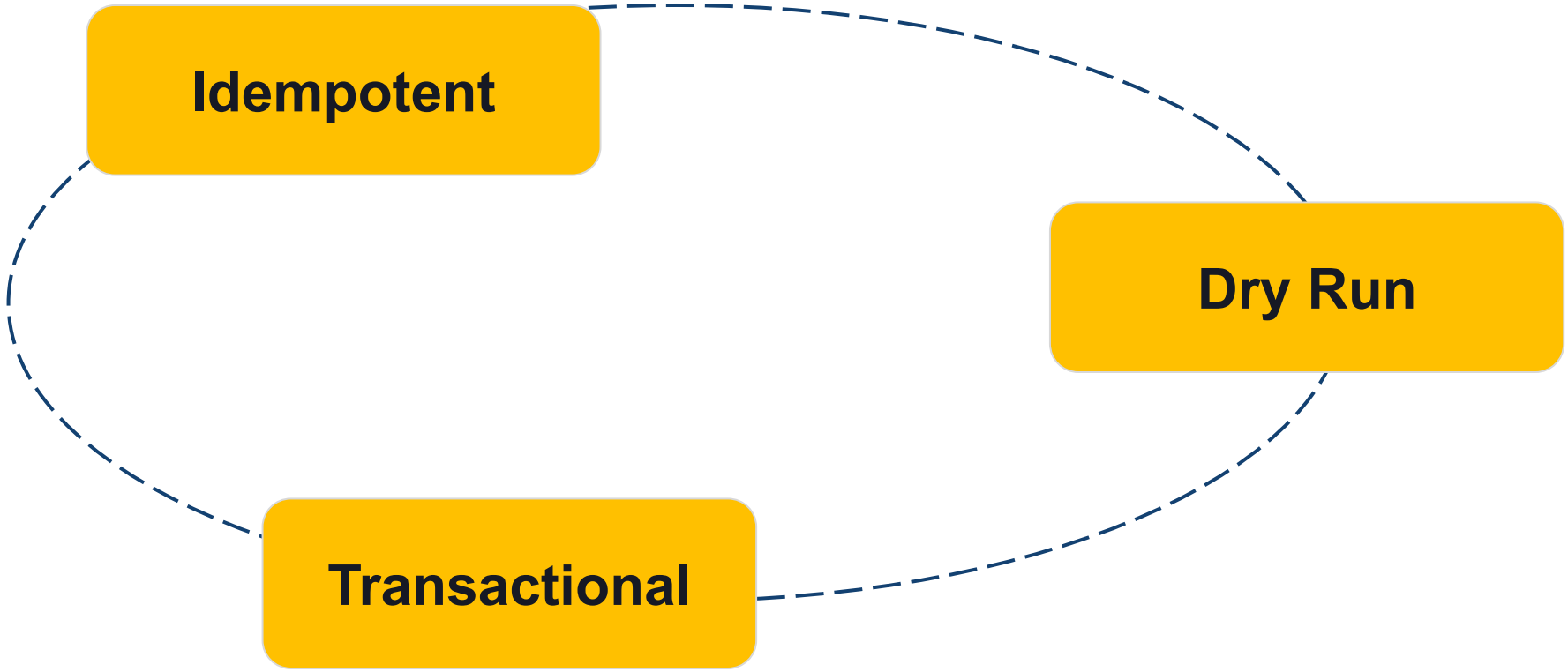
**Main Principles**

- Predictable
- Manageable
- Transparent
- Simple
- Reliable
- Human Friendly

**Design Principles**

- Idempotent
- Dry Run
- Transactional

# Virtuous circle of Design Principles



# Tools and Technologies that enable Trustworthy Automation

# Tools and Technologies to build Trust

**Main Principles**

- Predictable
- Manageable
- Transparent
- Simple
- Reliable
- Human Friendly

**Design Principles**

- Idempotent
- Dry Run
- Transactional

**Tools and Technologies**

- Declarative Vs Imperative
- Version Control
- Testing

# Declarative vs. imperative

## Imperative

**HOW**

Focuses on actions

## Declarative

**WHAT**

Focuses on outcomes

# Declarative Vs Imperative

```
configure terminal
interface GigabitEthernet0/1
switchport access vlan 10
exit
Exit
write memory
```

## Imperative - HOW

- Manually describe the step-by-step recipe.
- If something goes wrong halfway, state may be inconsistent.

**Focuses on actions**

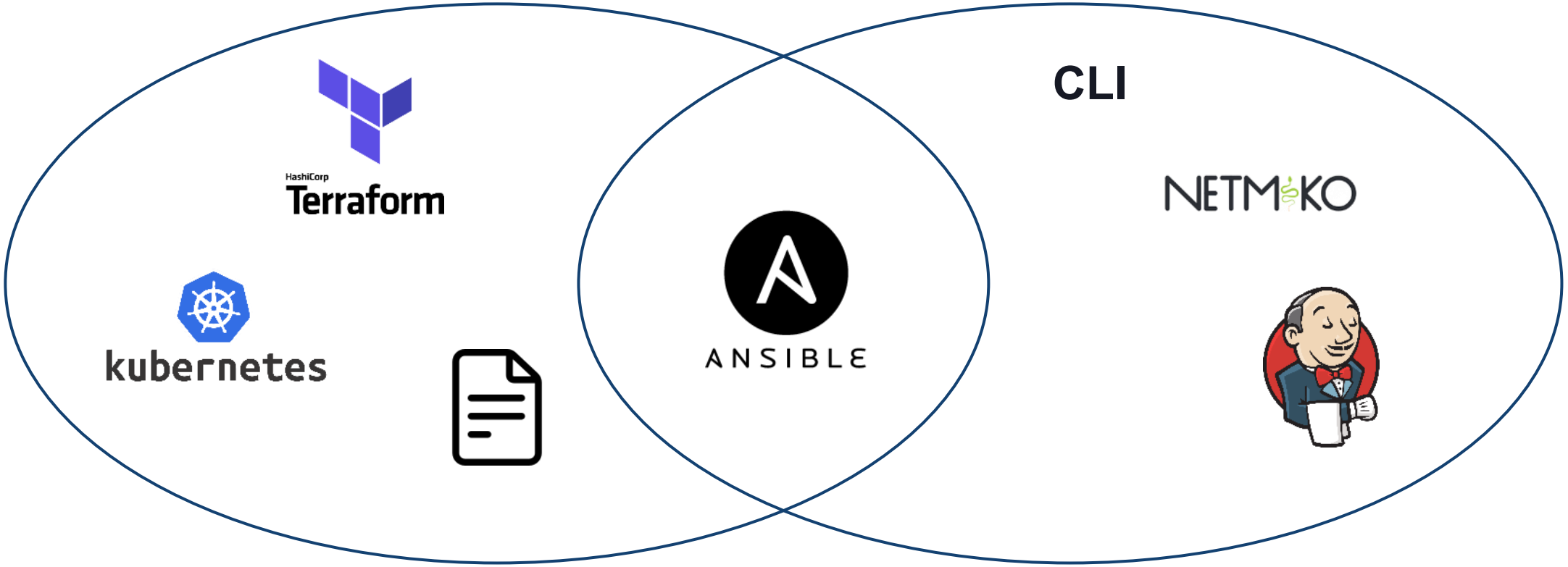
```
interface:
  name: GigabitEthernet0/1
  vlan: 10
```

## Declarative - WHAT

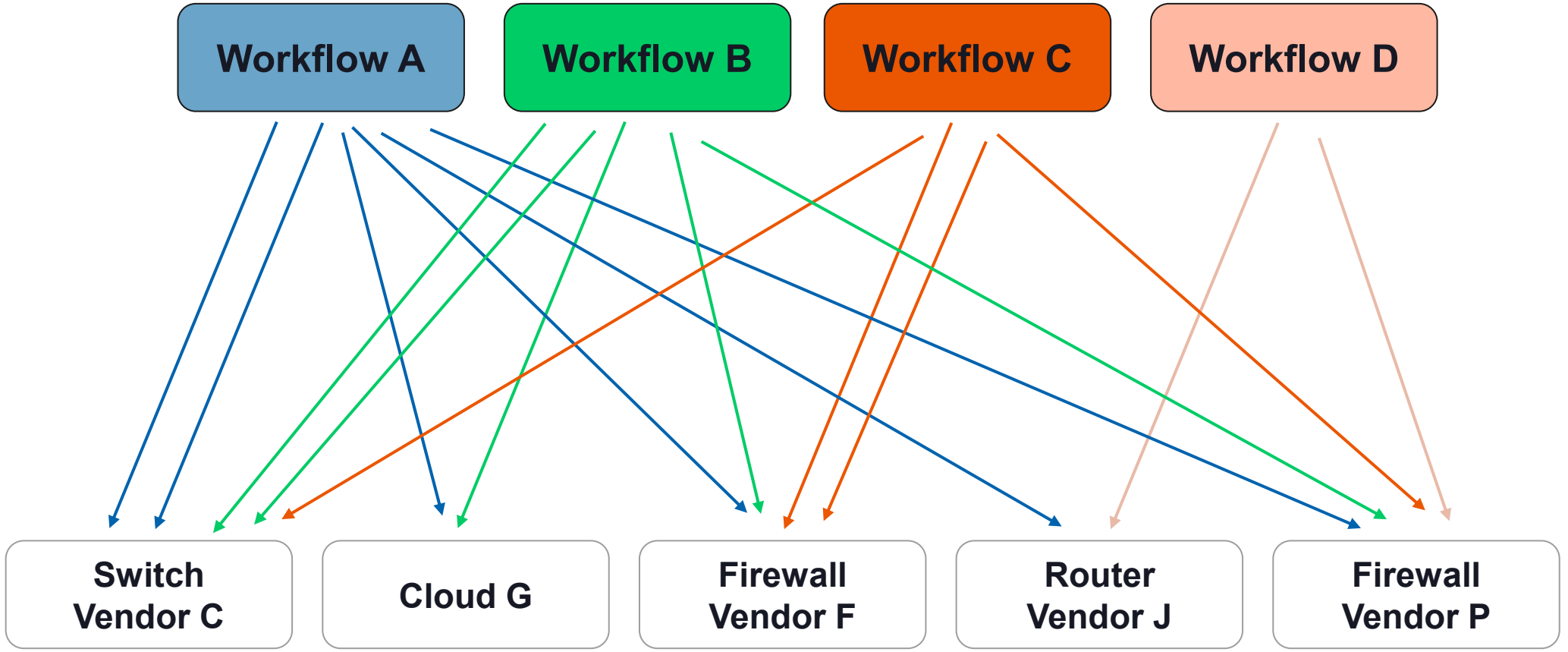
- You describe the desired end state, not how to get there.
- Easier to make idempotent and retry safely.

**Focuses on outcomes**

# Declarative Vs Imperative

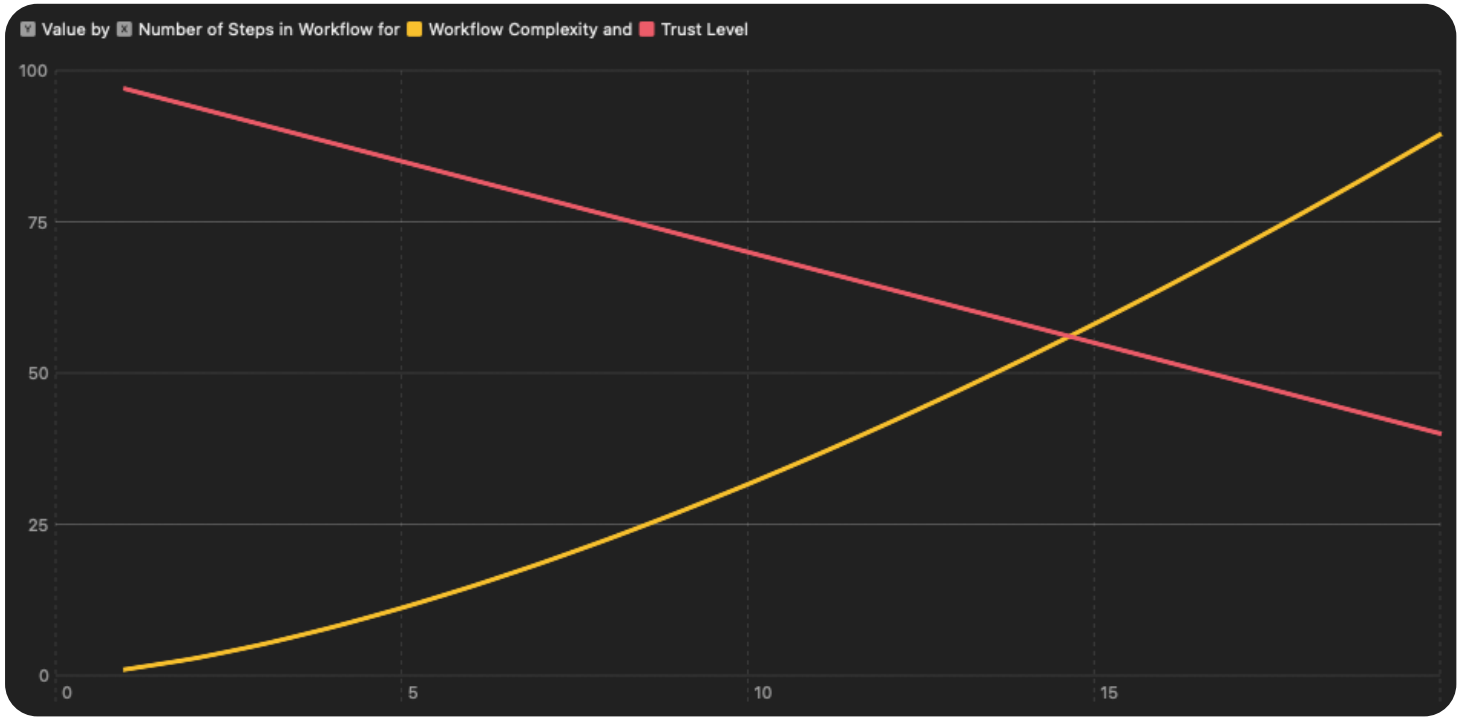


# Imperative Method

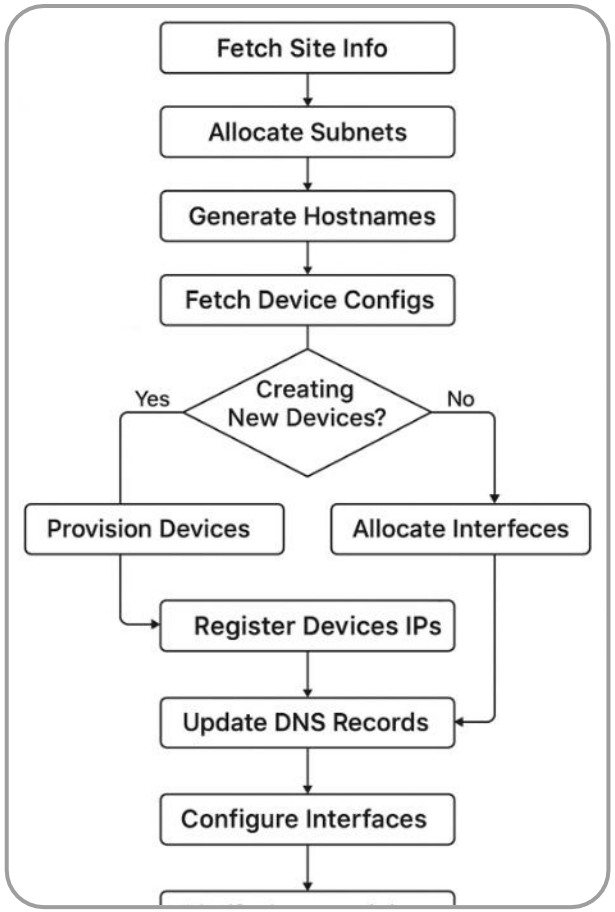


# Declarative Vs Imperative

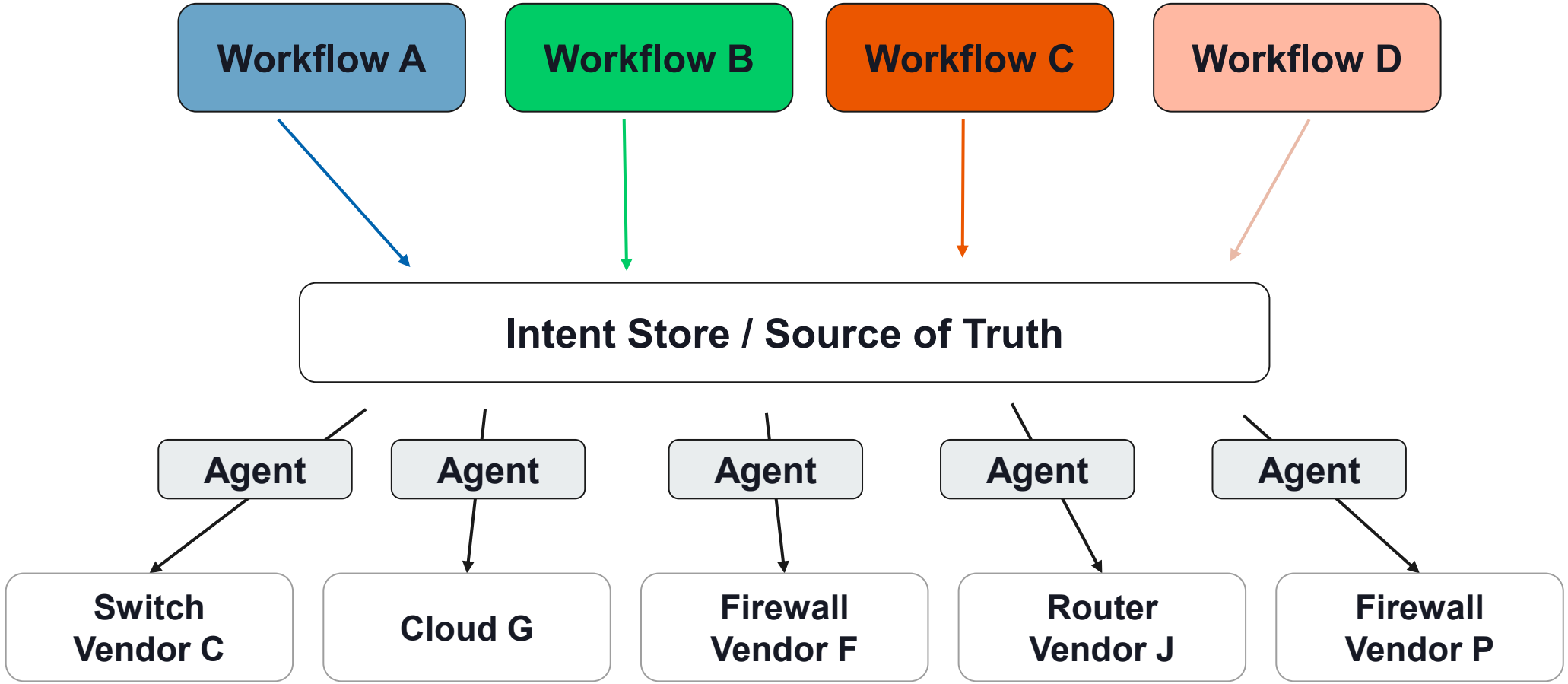
Imperative workflows are composed of multiple steps,
the more steps, the higher the complexity




**Number of steps in a workflow**



# Declarative Method



# Comparison with Design Principles

| | Imperative | Declarative |
|---|---|---|
| **Idempotent** | Hard | Easy |
| **Dry Run** | Hard | Easy |
| **Transactional** | No | Easy |

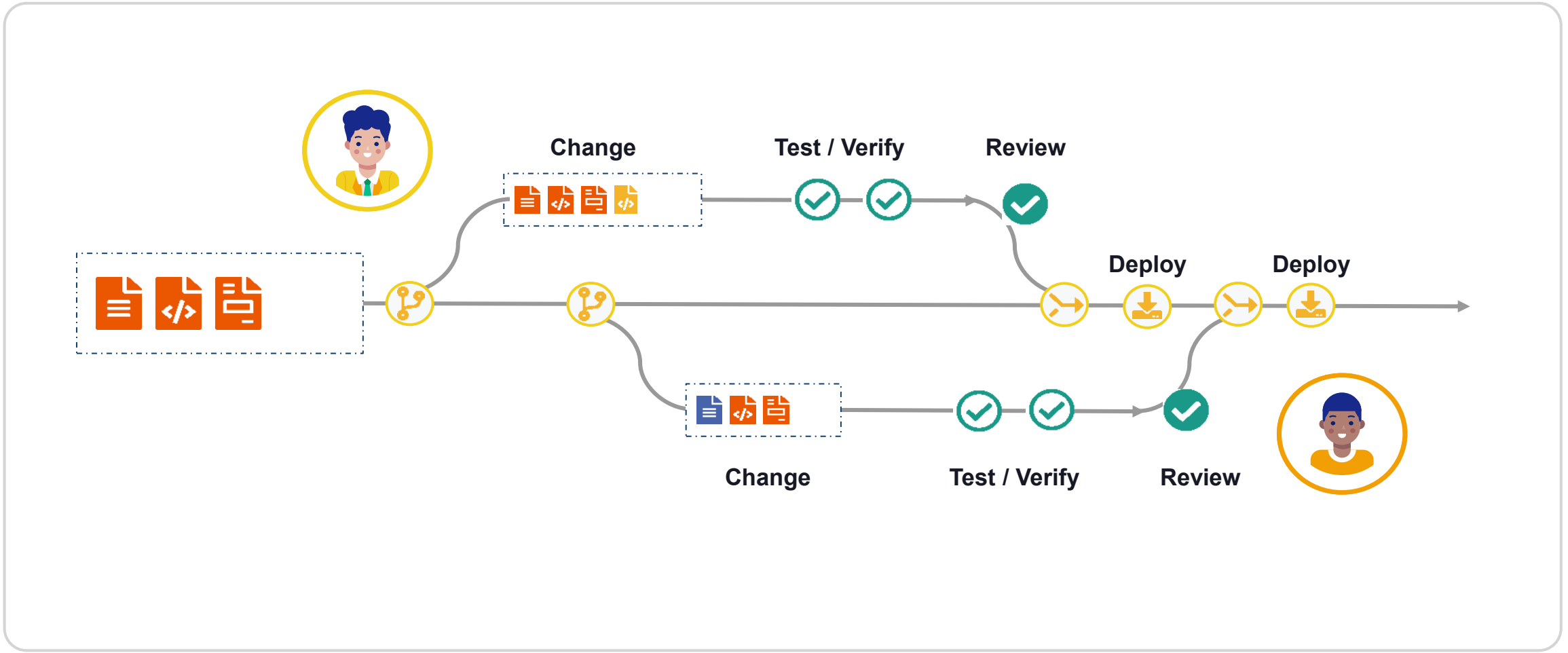
# Version control

Version control allows changes to be:

- Prepared in isolation
- Safely validated
- Reviewed

and only then integrated into the main automation environment.

# Changes are done in a branch



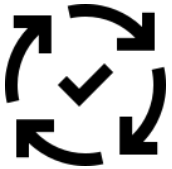
# Main benefits of Version Control

**Auditability and Traceability**

- See who changed what, when, and why.
- Essential for post-mortems and compliance
- Makes operations more transparent and safe

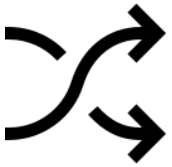
**Collaboration and Review (Change Management)**

- Team members can propose changes via PR
- Prevents risky or unreviewed changes from being pushed directly into production.



**CI/CD Pipelines**

- Automation workflows can be triggered automatically
- Changes can be tested and validated automatically before being deployed



**Atomic changes**

- Changes are grouped and committed as a single unit.
- There is no “partial change” state

# Comparison with Design Principles

| | Version Control |
|---|---|
| **Idempotent** | Easy |
| **Dry Run** | Built in |
| **Transactional** | Built in |

# Testing

Testing pushes you to design applications and workflows that are modular, observable, and deterministic.

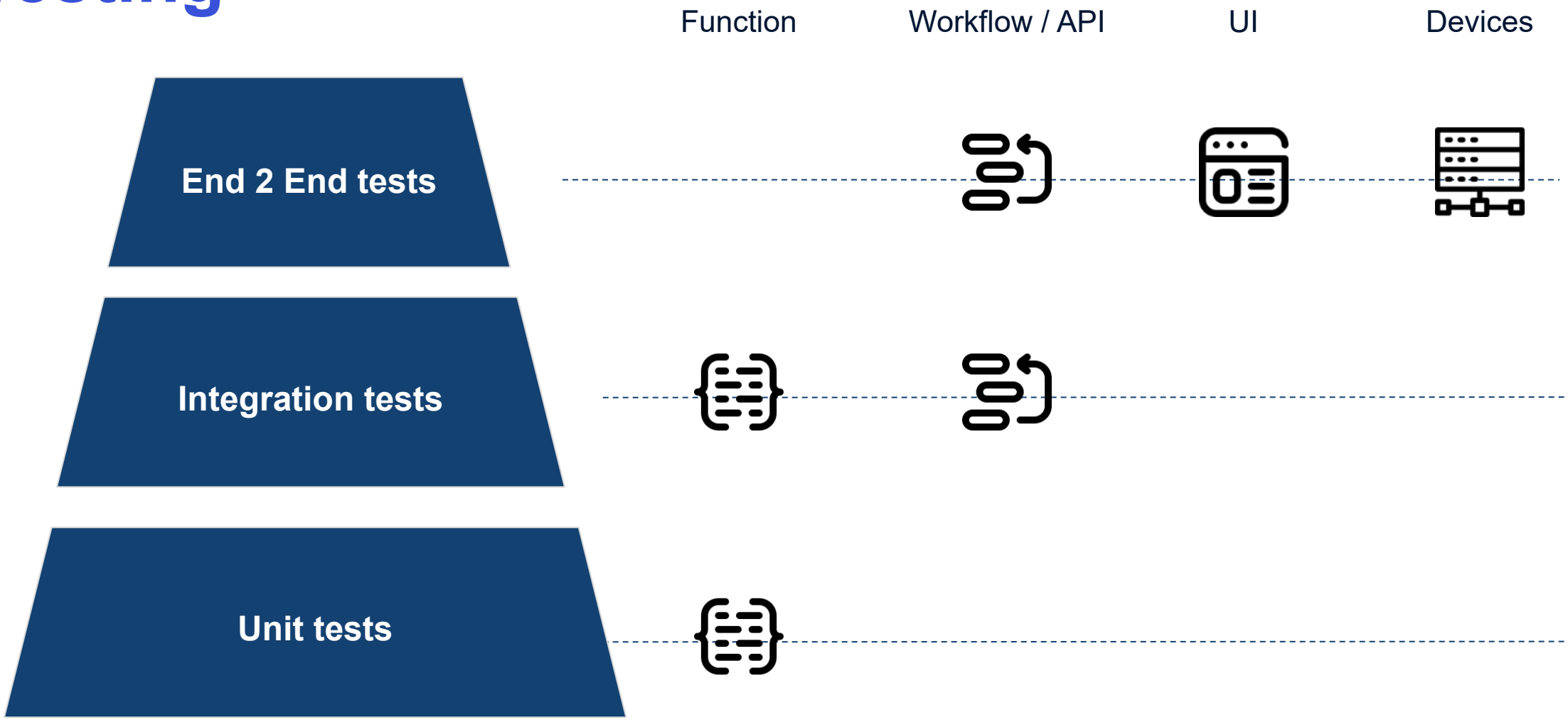
It encourages clear boundaries, clean inputs and outputs, and repeatable behaviors.

**Testable systems are a design choice.**

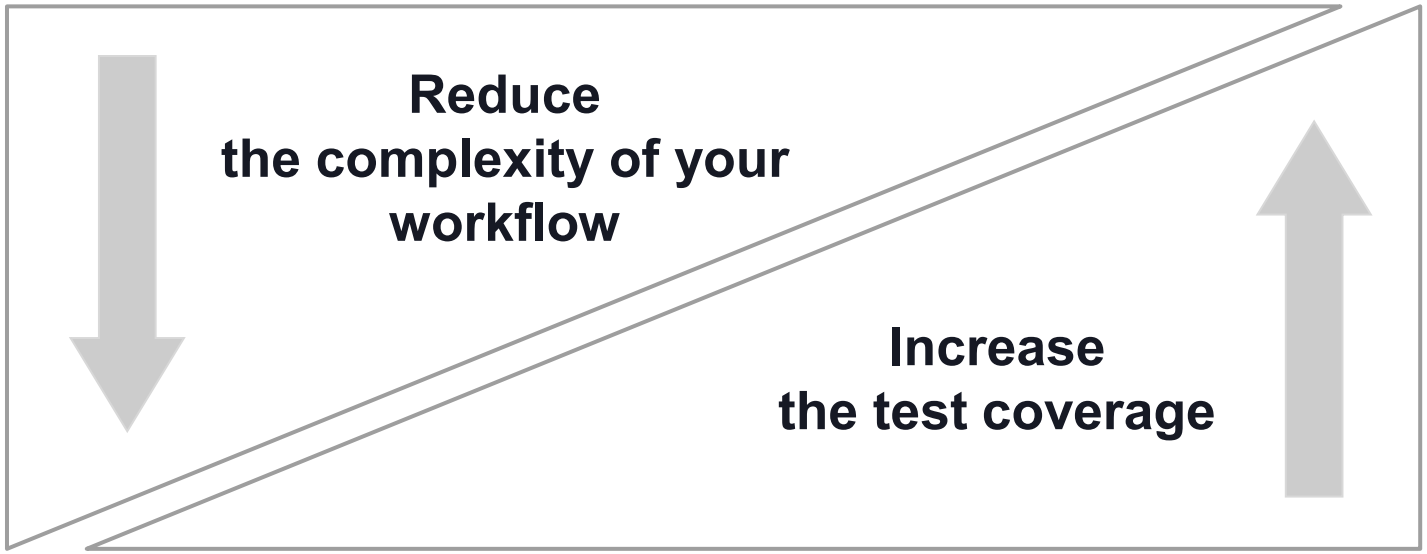
# Testing can be your superpower or your kryptonite

- As the complexity of the project increase, investment in testing are paying off exponentially
- Testing will become your development environment
- Proper tests allows to refactor with confidence
- Too many tests early on can be a burden to manage and slow things down

# Testing


Function
Workflow / API
UI
Devices
End 2 End tests
Integration tests
Unit tests

# Automation workflow testing

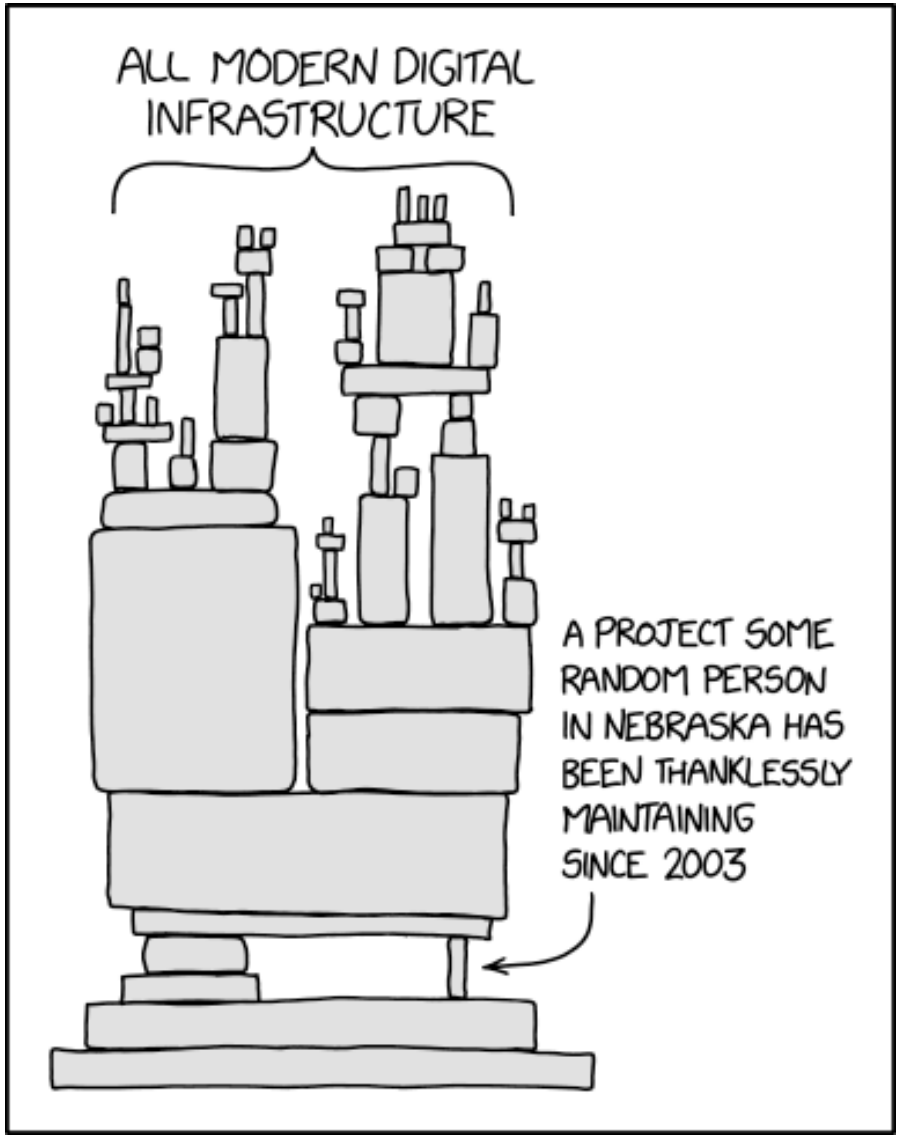




Write integration tests

Spend hours manually testing my workflow and still missing stupid bugs

# Practical Patterns for Building Trust




**Integrate what you CAN**

**Build what you MUST**

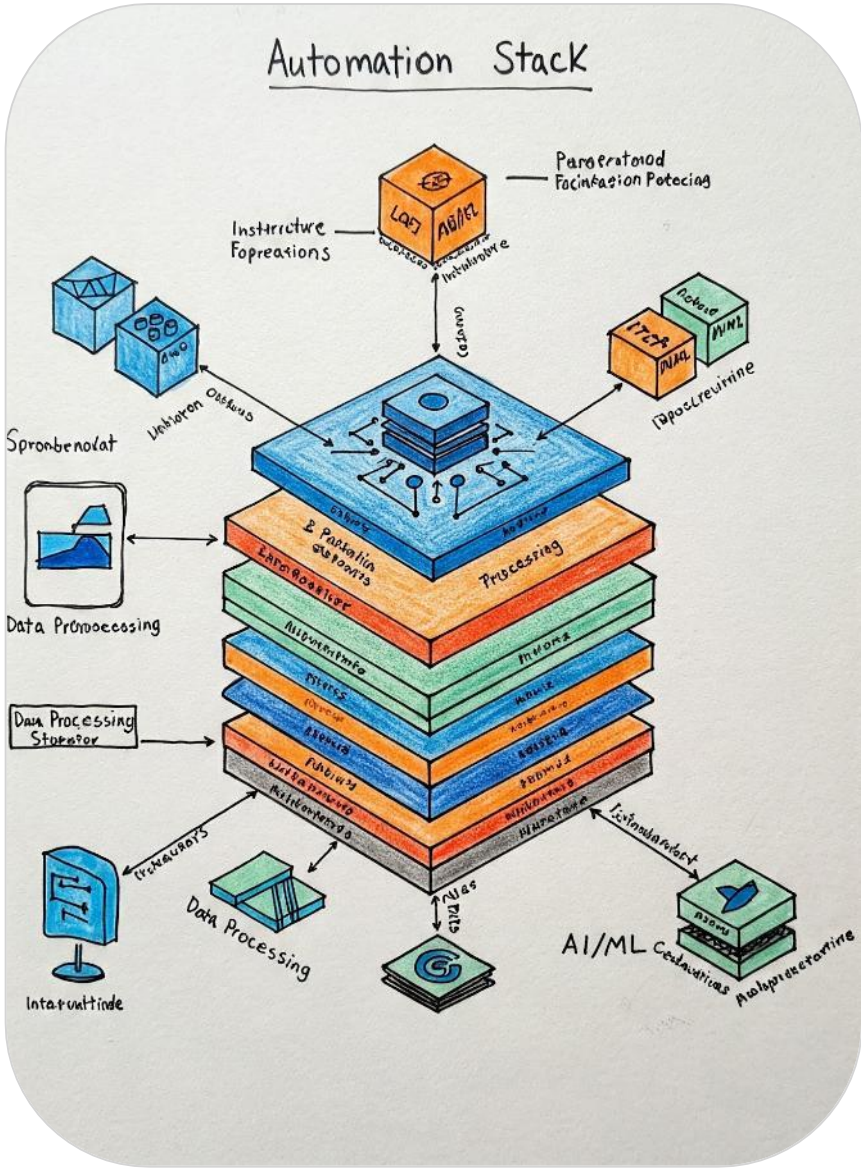
FRANCE-IX TOUR

15 ANS ÉDITION SPÉCIALE

# Select the right stack

Ensure the libraries / tools you are dependent on provides

- Programmable interfaces
- Declarative behavior
- Idempotency
- Test friendly interfaces
- Developer Experience
- Traceability & Logging



# The 3 primary attributes, classify your data

| | |
|---|---|
| **Role** | Capture the primary function of an object |
| **Status** | Capture all the stages of the lifecycle of an object |
| **Kind** | Capture the nature of an object |

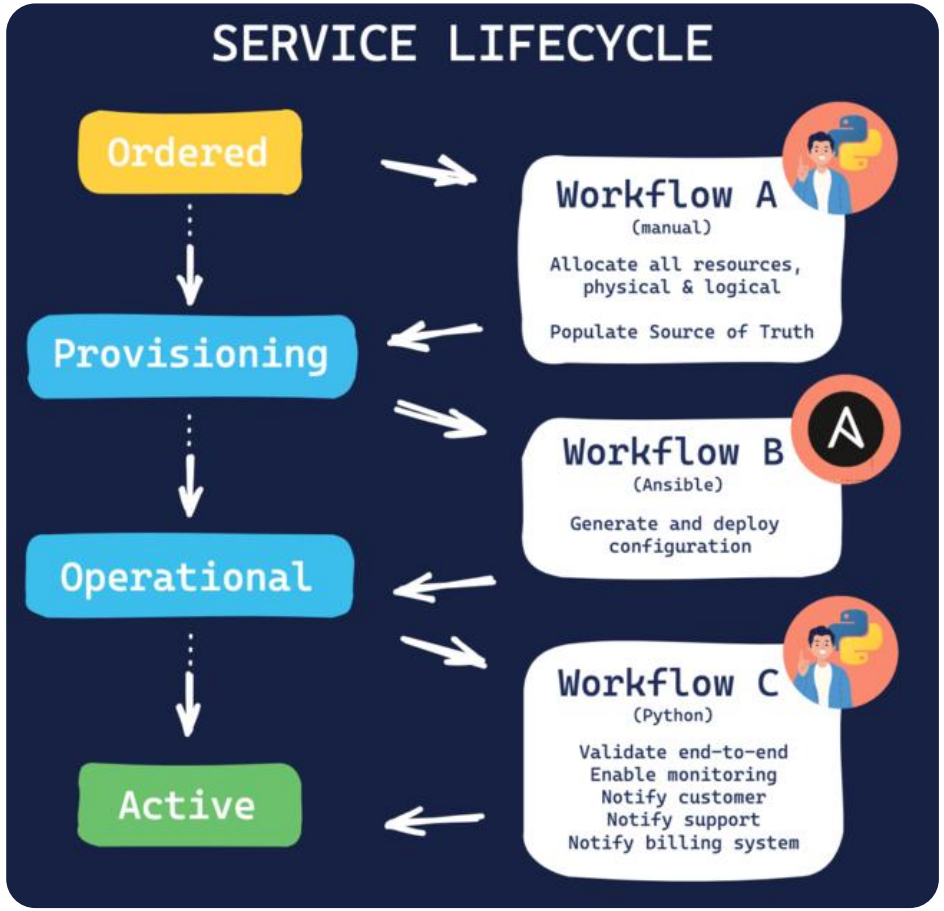
# Enforce business processes as part of your automation workflows

Maintenance windows are designed to ensure that no disruptive actions will be applied during business hours.

Similar rules should be embedded directly within your playbook

Ideally filter the valid target devices at the inventory level

- Only arista devices
- that are in maintenance mode



# Enforce business processes as part of your automation workflows

## Option 1 - Limited Inventory

```
---
- name: "Upgrade Software image on Arista Devices"
  hosts: platform_arista:&status_maintenance
  gather_facts: false
  tasks:
    - name: "Upgrade Software image"
      ...
```

## Option 2 - Inline Validation

```
---
- name: "Upgrade Software image on Arista Devices"
  hosts: platform_arista
  gather_facts: false
  tasks:
   - name: "Validate if the device is in maintenance mode"
     meta: "end_play"
     run_once: true
     when:
       - "device.status != 'maintenance'"
```

# Any questions?
# Thank You!


## Damien Garros - Co-founder & CEO
## OPSMILL


https://fr.linkedin.com/company/france-ix

https://x.com/ixpfranceix