

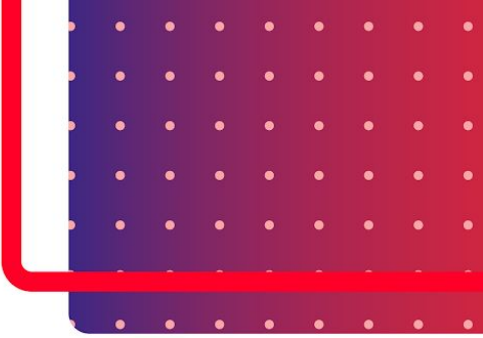


Connect

The Architecture of Intent

Why Kubernetes is a mindset not a tool

Joseph Baxter
Solution Architect



Agenda:

Kubernetes' Design Philosophy
Complexity Management
Extending The Kubernetes Core
Meta-Regulation of a System



Imperative vs. Declarative

Why imperative procedural automation hits an unsustainable scale wall

Imperative (The "How")

Step-by-Step Scripting & Runbooks

- **Executes Steps, Not Outcomes:** Focuses on sequential instructions rather than enforcing target state.
- **Static Assumption:** Assumes an ideal, unchanging world where every step succeeds without external interference.
- **The Complexity Scale Wall:** Each edge case requires custom branching logic. Code complexity explodes exponentially ($O(N)$ growth), making scripts fragile, unmaintainable, and prone to silent failures.

Declarative (The "What")

Intent-Based State Management

- **Declares Target State:** Users define *what* the desired result should be, hiding execution details.
- **Continuous Convergence:** Automated control loops constantly observe real-world state against intent and rectify drift.
- **Infinite Scalability:** Absorbs operational complexity within the framework. Adding edge cases or scaling nodes requires zero changes to the underlying model.



Governors

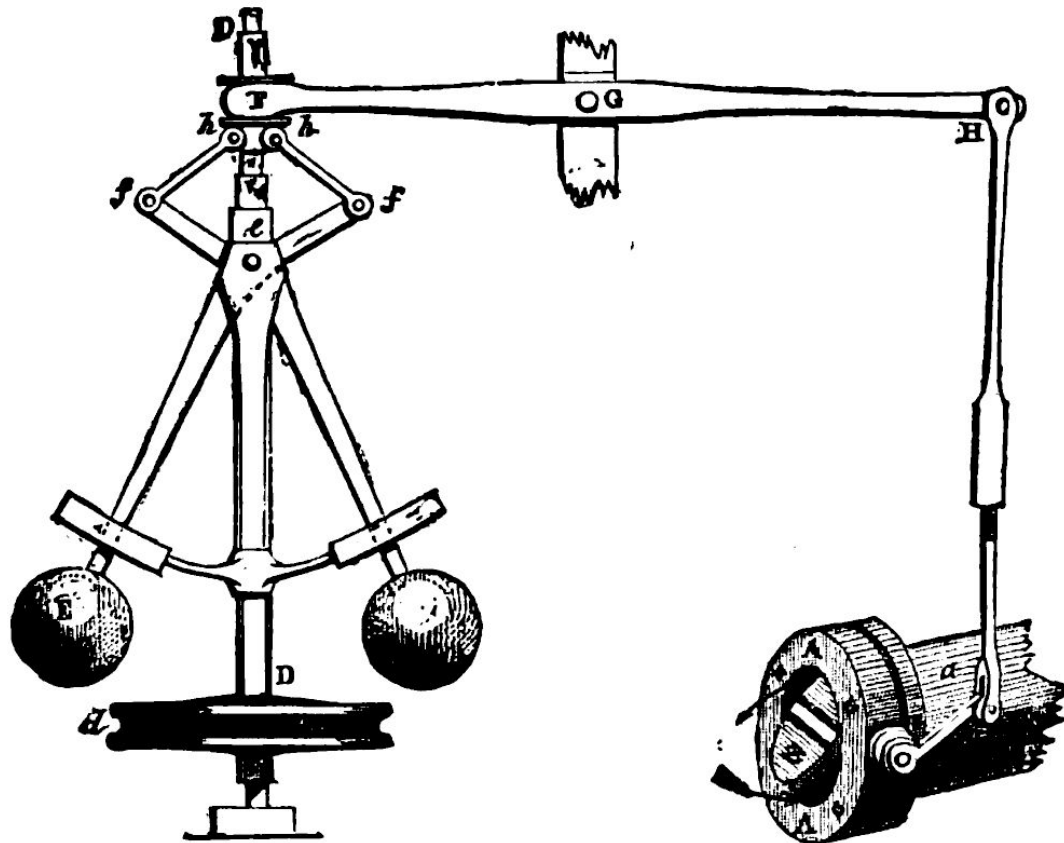
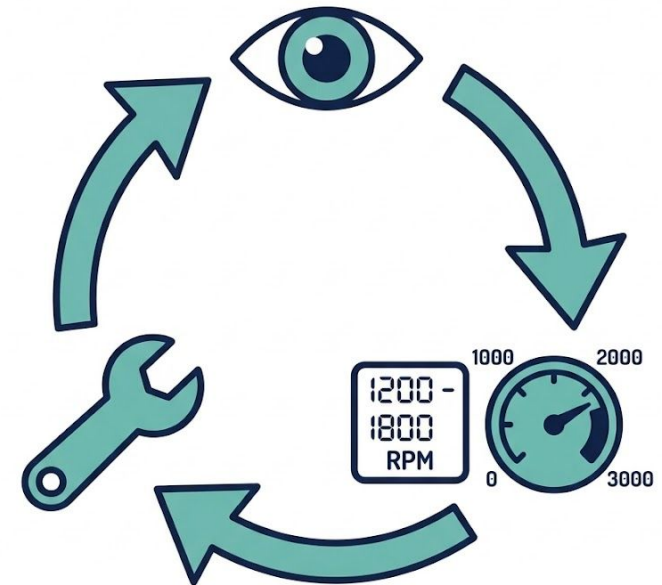
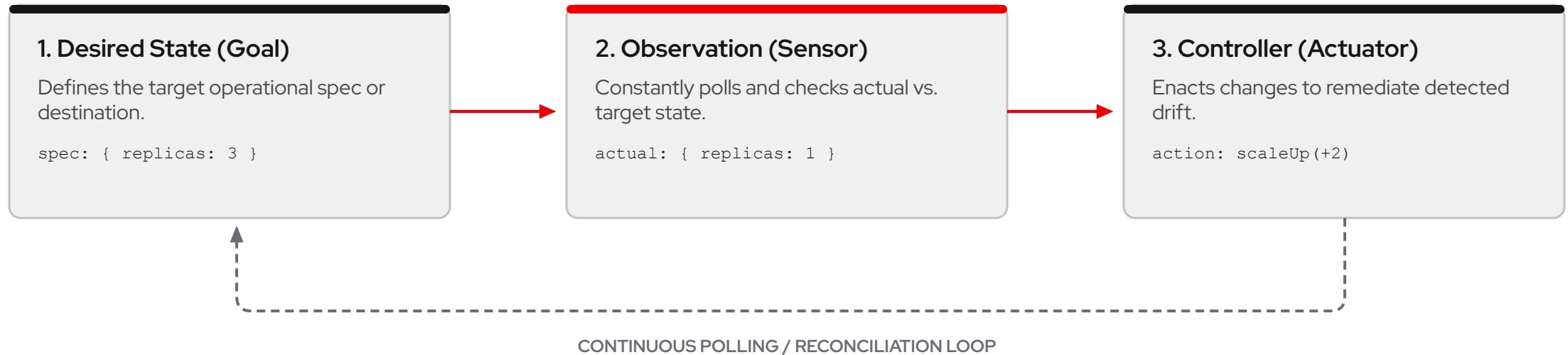


FIG. 4.---Governor and Throttle-Valve.



The Cybernetic Control Loop

Continuous reconciliation of actual state against target state



Constant Reconciliation

- **State Drift is Inevitable:** External failures, system chaos, or environment shifts will push the system away from its target state.
- **Self-Healing Mechanism:** By constantly measuring $\text{delta} = (\text{Desired State} - \text{Actual State})$, the controller dynamically applies corrective action without human intervention.
- **Declarative Abstraction:** Users only specify the *Goal*; the feedback loop absorbs all runtime complexity.



Kubernetes Boundary Problem

Vanilla Kubernetes manages containers, but treats the host, network, etc. as unknowable chaos.

Inside Boundary: Kubernetes Scope

Pods

Smallest deployable units executing container workloads.

Deployments

Declarative state management for pod replicas.

Services

Abstractions for pod networking and internal load balancing.

Ingress Controller

Manages external HTTP/HTTPS routing into internal services.

Outside Boundary: Unmanaged Factors (Disturbances)

Host OS

Kernel configuration, driver updates, and host storage state.

Host Status

CPU/Memory pressure, disk I/O bottlenecks, hardware health.

Systemd

Host init daemon, system unit states, and non-containerized services.

Network Infra

Underlying host interfaces, routing tables, and physical switching.



Managing Kubernetes Infrastructure

Extending environmental variance management forces complexity elsewhere

Kubernetes Orchestration

Sleek, Managed & Simple

- **Declarative Intent:** Define target state and let native controllers reconcile drift automatically.
- **In-Loop Control:** Self-healing container workloads operating within clear scope boundaries.
- **Standardized Abstraction:** Clean API objects insulating developers from runtime noise.

Supporting Infrastructure

Ad-Hoc Automation & 3 AM Pages

- **Out-of-Loop Failures:** External disturbances (Host OS, network, hardware) lie outside core control loop.
- **Toil & Remediation:** Handled via manual Admin fixes, SRE teams, or fragmented Event-Driven Automation.
- **Dual Systems Burden:** Must maintain both the K8s system AND the supporting host system.

The Toil & Remediation Problem

- **Absorbing Variance:** To expand system capability and handle environmental drift, complexity must be absorbed elsewhere in the stack.
- **External Disturbances:** When disturbances cause failure outside the control loop, fixing them introduces secondary remediation efforts.
- **Two Systems to Manage:** Operators are left supporting two distinct environments—the Kubernetes cluster and the underlying support infrastructure.



Ashby's Law of Requisite Variety

Only Variety can Absorb Variety

$$V_R \geq V_E$$

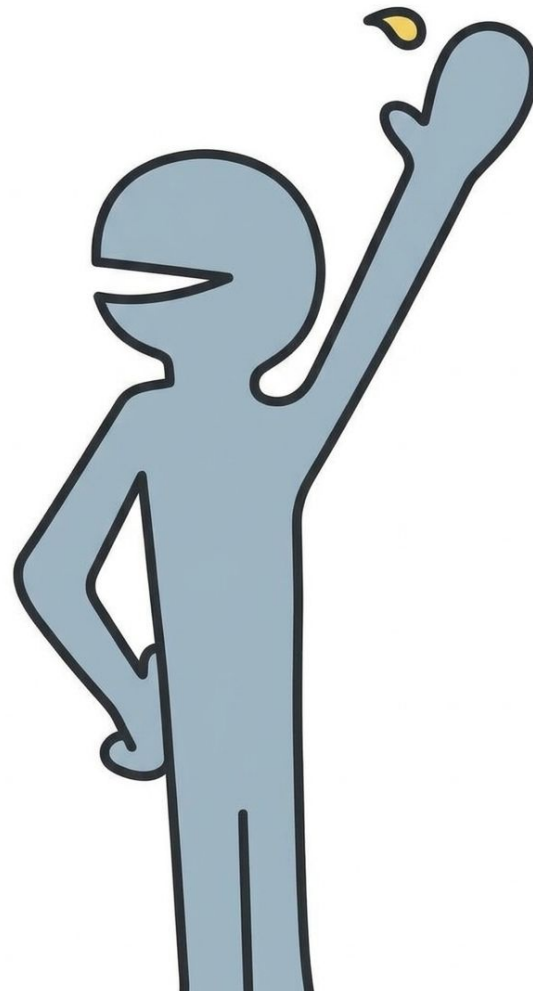
Where Variety represents the # of possible states in a system.

In order for a system, that operates in a complex environment, to seem simple; it must be very complex.



Where does Excess Variety Go?

Biological variety absorption



When the System can't handle an external disturbance it falls on the Engineers to fix.

In Kubernetes 1.25 - who had to implement PodSecurityPolicies?



Extending the Kubernetes Control Landscape

Using the Operator pattern to integrate custom domain logic and external infrastructure into the control loop

Custom Logic & Operators

Extending the API

- **Custom Object Definitions:** Introduce CRDs into the native Kubernetes control plane.
- **Dedicated Controllers:** Embed domain-specific operational knowledge directly into code.
- **Native Integration:** Objects maintain their own state configuration within core loop.

Expanded Scope Capabilities

Platform & Host Control

- **Host Machine Control:** Manipulate system configurations, kernel, and host state.
- **Dynamic Networking:** Reconfigure routing rules and physical/virtual network layers.
- **Complex Services:** Automate deployment of databases and observability stacks.

In-Loop Management

Automated Self-Healing

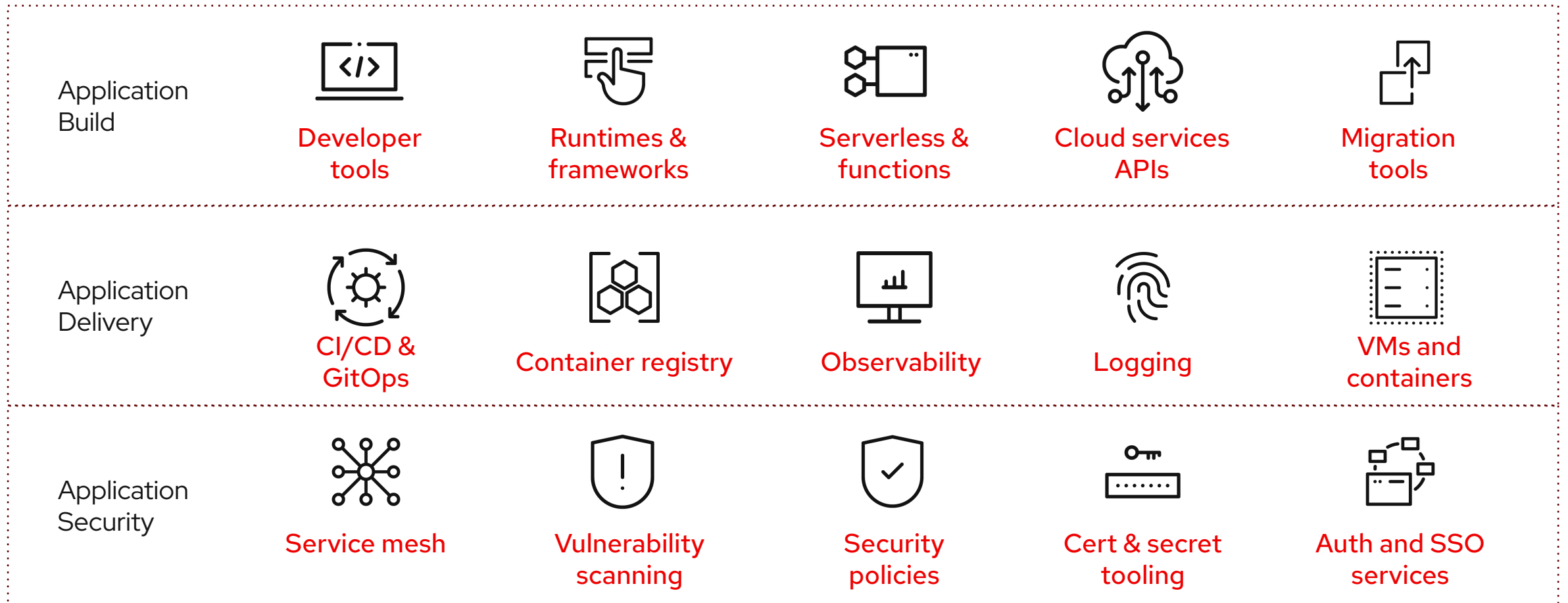
- **Continuous Audit:** Actively monitor actual vs. desired state across all layers.
- **Automated Remediation:** Fix external drift and disruptions without manual human toil.
- **Unified Operations:** A single declarative model for both workloads and infrastructure.

Core Paradigm Shift

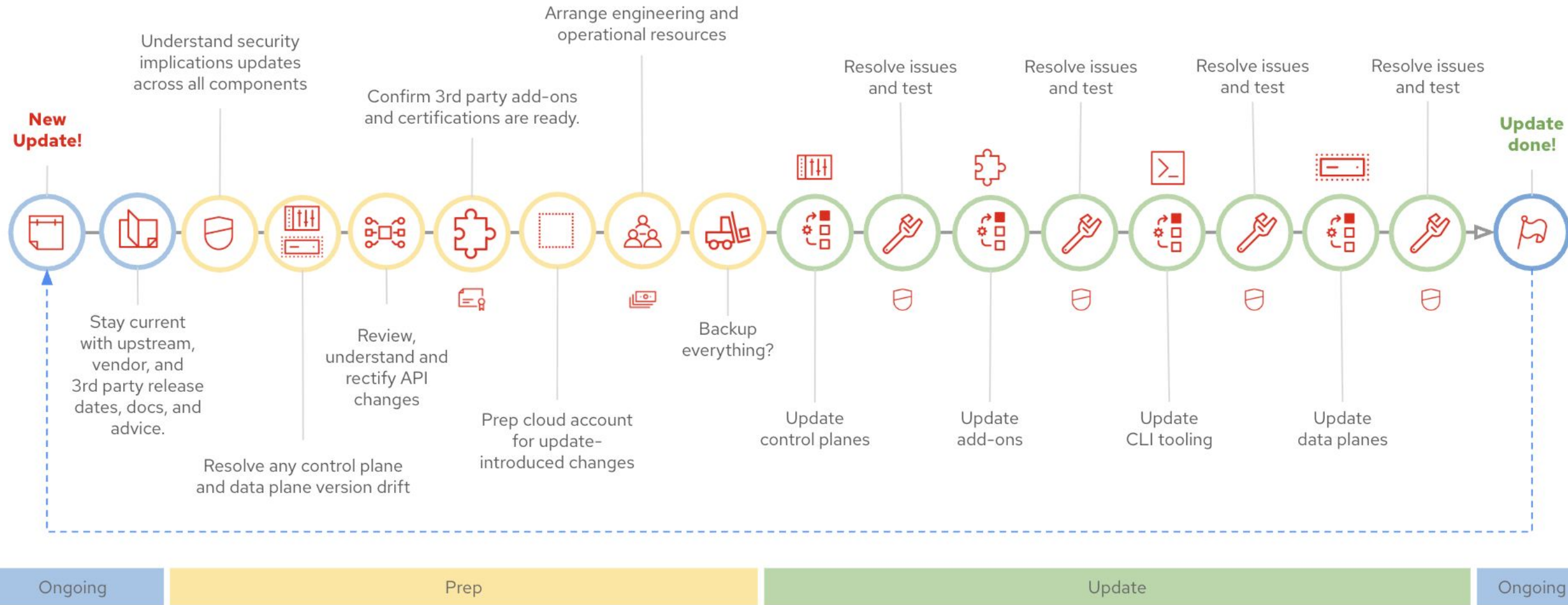
- **Eliminating Toil:** External disturbances (hosts, networks, storage) are brought inside the reconciliation loop rather than handled by out-of-band automation.
- **Combined Benefits:** Delivers both rich feature functionality and continuous self-healing remediation across the entire stack.



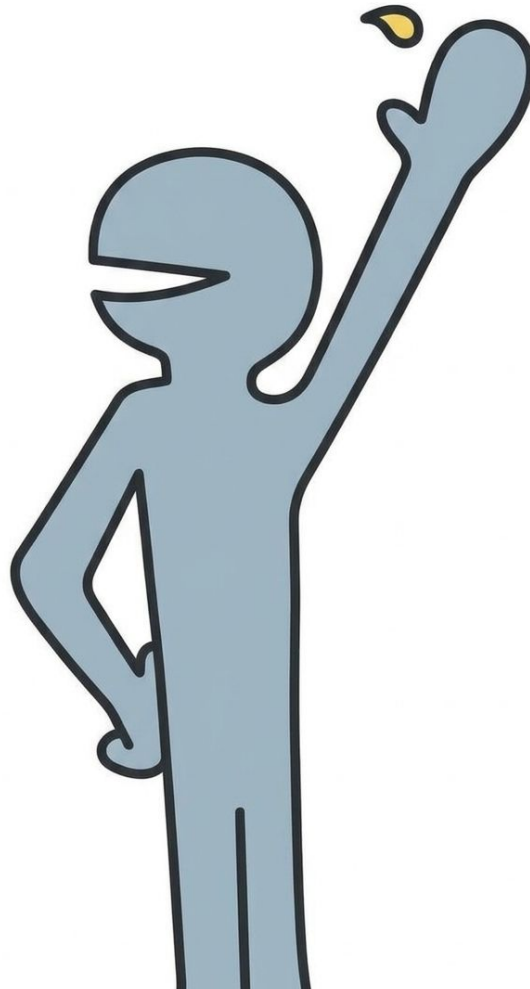
Components of an Application Platform



Mapping the Update Flow

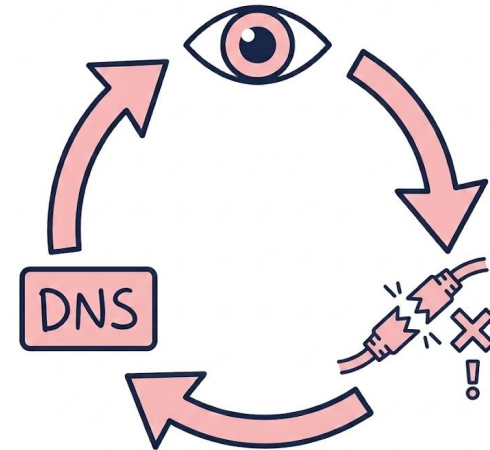
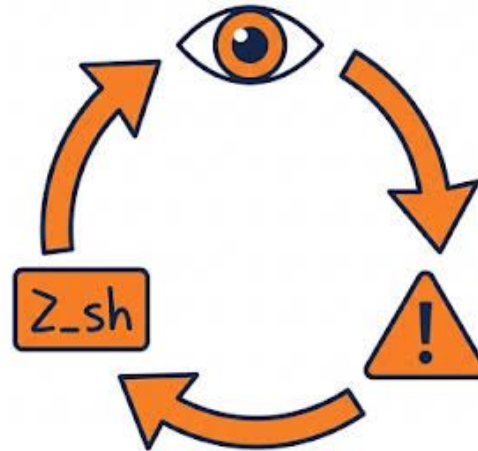
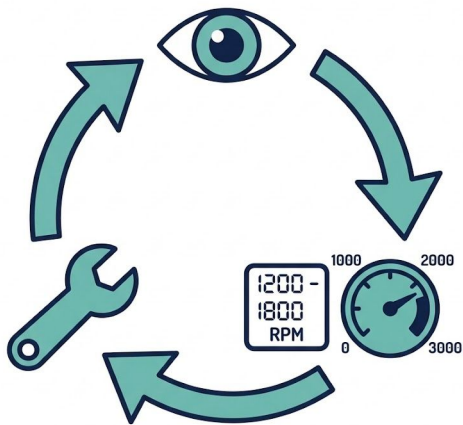


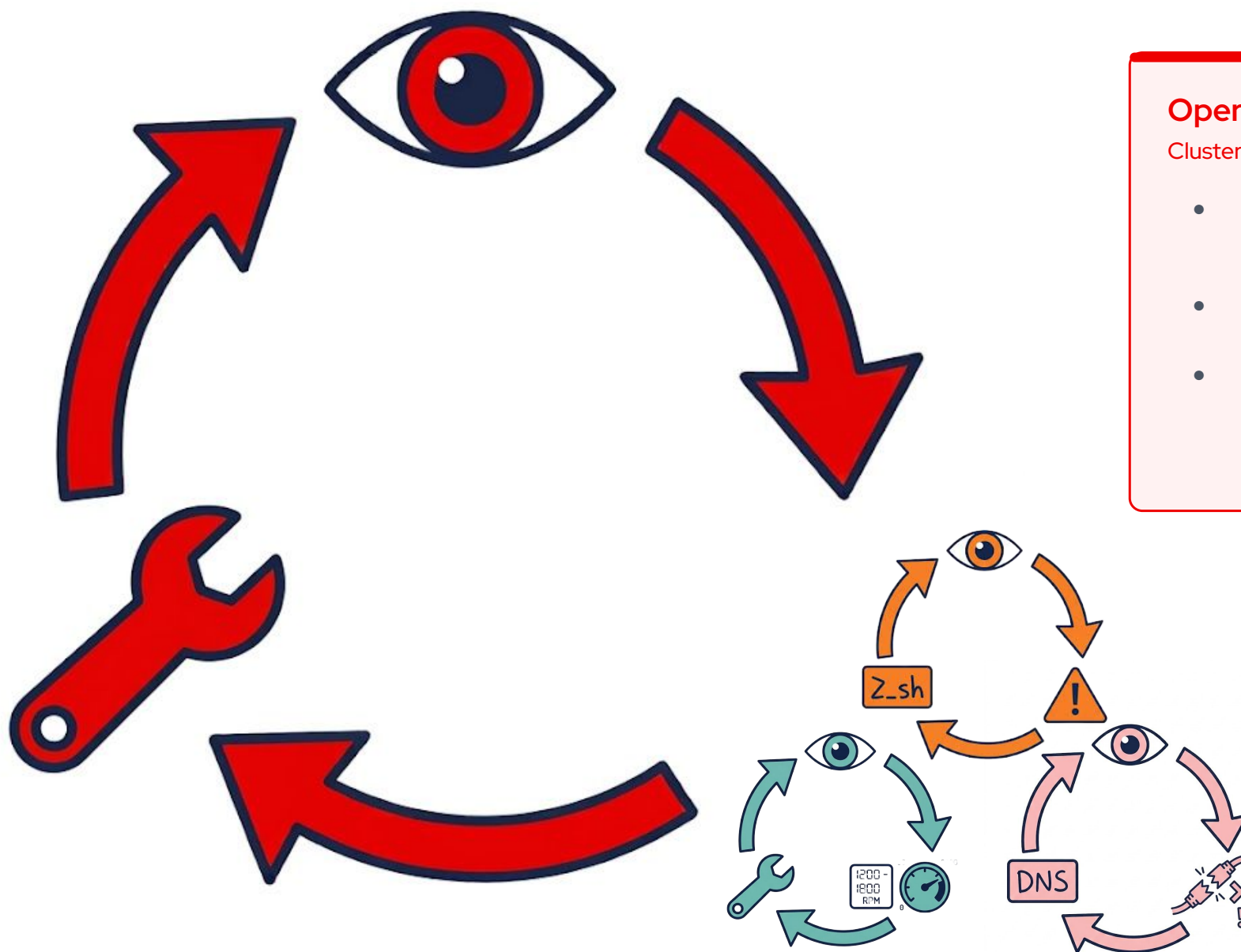
Where does Excess Variety Go?



2nd Order Cybernetics

Who Regulates the Regulators?





Operator of Operators

Cluster Version Operator

- **Tested Dependency Matrix:** Aware of underlying operator compatibility and dependencies
- **Self-Regulating:** Considers itself part of the system
- **Intelligent Assessment:** Can actively search for breaking config and restore cluster state to previous versions

Closing Insights

Architecting for Scale & Systemic Absorbption

01

Cybernetic Loop

Kubernetes is built around a core cybernetic loop, where **autonomy** and **abstraction** allow for immense scale.

...

02

Absorb Variety

If you're going to introduce variety to your system, you need to architect that system to **absorb that variety**—not pass it on to engineers.

...

03

Extend the Loop

By extending the core cybernetic loop of Kubernetes, you can maintain a **unified model** of the entire platform and handle systemic complexity seamlessly.



Connect

Thank you



linkedin.com/company/red-hat



facebook.com/redhatinc



youtube.com/user/RedHatVideos



x.com/RedHat

