Authority before automation

An executable control model for intent, observation and authority transfer

Jeleel Muibi · HybridOps.Tech · Version 1.1 · August 2026

Abstract

Infrastructure automation can execute the wrong decision correctly. An address, network, machine identity or service relationship may exist simultaneously in an intended-state system, an infrastructure state file, a provider API and live observation. Those records carry different responsibilities and authority. Automation consumes the record explicitly authorised for the governed fact, with freshness serving as supporting evidence rather than an ownership rule.

This paper introduces an executable authority model for that boundary. Each governed fact has one declared authority at a time; intended state, implementation specification, execution binding and observed condition remain separate records; and dependent automation proceeds when the required authority is selected and ready. Bootstrap and emergency operation are explicit authority transitions in which temporary or observed information can be qualified before it becomes durable intent.

The model is implemented in HybridOps through authority-aware operation contracts, readiness gates, required publication and separate live verification. The worked path establishes a bounded NetBox bootstrap, transfers addressing authority to the intended-state service and then requires downstream infrastructure to consume governed IPAM.

The operational question is:

Which record is allowed to decide this fact before automation acts on it?

1. The core failure is ambiguous authority

Hybrid estates often contain several records describing the same fact:

- a design or intended-state record;
- an implementation declaration;
- infrastructure state binding that declaration to provider objects; and
- a live observation from the running environment.

The records serve different purposes. A cloud API reports whether an instance exists; intended-state authority decides whether that instance belongs in the approved design. Infrastructure state binds one operation to provider objects; the organisation's address plan remains with its declared authority. Monitoring reports current service condition; desired topology remains an intent decision.

When these roles are implicit, automation tends to adopt one of two weak rules: use the newest value or use whichever system the current script can query. Both convert implementation convenience into policy.

The authority problem is therefore precise: for each governed fact, which record is permitted to supply intent, and what must be true before a dependent operation may consume it?

Established approaches and the remaining gap

The separation between intended and operational state is well established. RFC 8342 distinguishes intended configuration from operational state and recognises that the two can diverge because of propagation, unavailable resources or system behaviour [1]. RFC 9144 describes those datastores as different viewpoints across the management lifecycle [2]. NetBox positions its model as intended network and infrastructure state

rather than automatic discovery [3]. GitOps and controller patterns establish a related principle: controllers reconcile toward declared intent, and ownership must be clear when more than one controller can affect the same object [8,9].

Those approaches establish the importance of desired state and ownership. The remaining cross-system question is how a bounded infrastructure operation selects one authority for a governed fact, proves that authority is ready, transitions away from bootstrap authority, publishes required outcomes back to the intended system, and requires an explicit transition before live observation becomes durable intent.

The innovation introduced here is an executable authority contract around that handoff. Authority is a runtime condition that can permit or block dependent work.

2. Four records, four responsibilities

The model keeps four records separate.

Record	Question	Responsible record
Intended state	What should exist and under which relationships?	Declared authority for the governed fact
Implementation specification	How should the outcome be produced?	Versioned operation contract
Execution binding	Which provider objects belong to this operation and environment?	Runtime and native infrastructure state
Observed condition	What exists now and is it usable?	Live provider and workload checks

Comparison between these records supports drift detection. Ownership transfer requires an explicit transition.

A live system may reveal drift. That observation creates a decision: restore the running system to intended state, or approve the external change and update the authoritative declaration. Promoting a live value to intent therefore requires deliberate authority change.

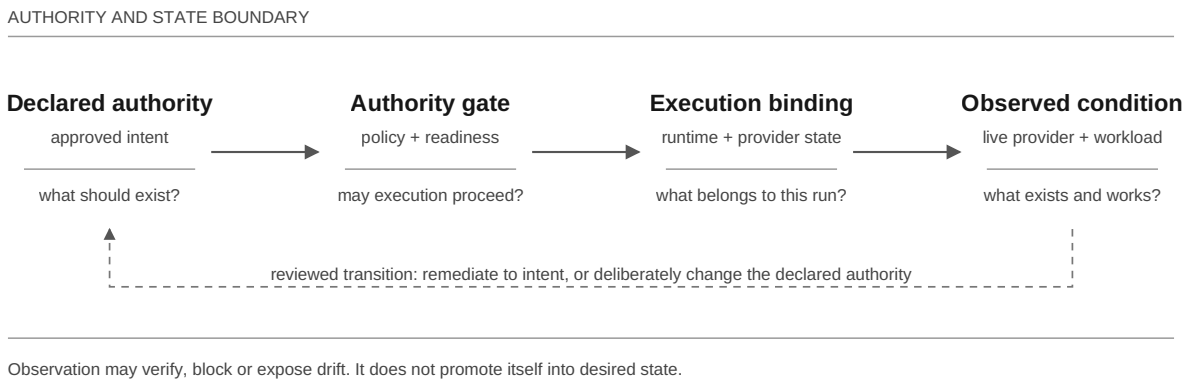

Figure 1. Intended state, execution binding and observed condition remain separate across an explicit authority boundary.

3. Authority selection is explicit ownership

External review of v1.0 sharpened the wording around "authority per fact". The runtime selects one declared authority for each governed fact and treats other records as implementation bindings or observations.

For a governed fact, the normal operating state is:

```
one declared authority
        |
        +--> automation consumes its approved intent

observed/provider state
        |
        +--> verifies, blocks or exposes drift
        |
        +--> requires explicit authority transition before becoming desired state
```

When two systems appear to own the same fact, the ownership conflict is resolved through explicit policy or operator transition.

This is particularly important as automation agents become more capable. An agent proposes or executes changes inside the authority and constraints it has been given. Durable intent therefore comes from explicit authority, while live state supplies observation and verification.

4. Make authority executable

An operation that consumes a governed fact states both the mode it needs and the authority it expects. The enclosing lifecycle makes the same decision. A disagreement is an invalid operation rather than a value to guess at runtime.

For an authority-dependent step, a practical gate establishes:

1. the lifecycle policy and dependent step agree on the required authority;
2. the authority environment and identity are valid;
3. required prior state for the authority exists;
4. required outputs needed by the dependent operation are available; and
5. when policy requires it, the authority is reachable through a live check.

A failed required condition stops dependent execution before infrastructure changes are made from guessed or stale values.

The same principle applies after execution. Where publication is part of the governed contract, governed completion includes successful publication of the required authoritative record.

5. Treat bootstrap as a temporary authority state

An intended-state service can create a circular dependency: infrastructure must exist before the authority service can run, while the authority service is expected to govern later infrastructure.

The model resolves the cycle by assigning bootstrap a bounded temporary authority scope.

```
bounded bootstrap input
        |
        v
minimum network and hosts
        |
        v
intended-state service ready
        |
        v
seed authoritative dataset
        |
        v
explicit authority transfer
        |
        v
dependent infrastructure consumes governed state
```

After transfer, downstream operations consume the governed authority. This creates a visible point at which ownership changes hands.

6. Emergency and brownfield changes use explicit authority transition

Brownfield and emergency operation provide the strongest test of an authority model. An outage may force an engineer to change a device, controller or provider directly. At that moment, declared intent may still represent policy while the live system reflects the immediate service-restoration action.

External review of v1.0 highlighted exactly this reconciliation window [14]. Declared intent, controller state and a freshly rediscovered device may all disagree at once.

The emergency path therefore uses a visible transition:

1. qualify the observed emergency state and the reason it exists;
2. decide whether it is temporary or should become the new desired state;
3. if temporary, restore the declared authority when the incident allows;
4. if permanent, update the authority deliberately and record the change; and
5. reconcile from the resulting declaration.

The exact approval mechanism is organisational policy. The runtime records the transition, and simultaneous ownership claims become an explicit conflict that requires a rule or operator decision.

7. Operation-scoped authority and continuous reconciliation

A continuously reconciling controller and an operation-scoped runtime solve different cadence problems.

Where one reconcile plane already owns the full resource lifecycle, that controller remains the natural authority and enforcement boundary. HybridOps applies its authority gate when a bounded operation crosses multiple reconcile planes or when a bootstrap, publication or handoff must complete before another system can continue.

A bounded authority-aware operation can therefore:

- refresh derived inputs from the declared authority;
- detect changed declarations or stale successful state;
- perform required live checks independently of historical run success;
- compare observed provider/workload condition with intended state; and
- require an explicit authority change before an external modification becomes new intent.

This is consistent with security-focused configuration management, where baselines, controlled change and monitoring remain separate responsibilities [6,7].

8. Worked authority transfer

HybridOps is the open-source implementation used to exercise the model. The worked path begins with a bounded bootstrap-netbox operation, establishes and seeds the intended-state service, and then moves downstream infrastructure to an authoritative-foundation operation whose dependent steps require governed IPAM.

Boundary	Expected behaviour
Bootstrap	Static inputs hold temporary bootstrap authority until transfer
Transfer	Lifecycle policy and dependent steps agree on the new authority

Boundary	Expected behaviour
Readiness	Missing state or a required failed live check blocks consumption
Publication	Required authoritative publication must succeed before governed completion
Observation	Provider/workload checks report running condition while intended state remains separately authoritative

NetBox is the current intended-state implementation. The innovation is the authority transition and executable gate around consumption and publication.

9. Failure conditions and evidence

The model fails closed when:

- lifecycle policy and step-level authority requirements disagree;
- required authoritative state is absent or unusable;
- the selected authority cannot be reached when live readiness is mandatory;
- required publication fails or produces an unusable record;
- two mechanisms claim incompatible ownership without an approved transition; or
- an operator or agent attempts to promote observed external state into durable intent without explicit authority change.

A governed run makes four facts inspectable after the change:

1. which record authorised the governed decision;
2. which versioned operation consumed it;
3. which provider resources were bound to the operation; and
4. which independent checks established the resulting condition.

Those records provide concrete proof of where authority entered the operation and how an exception or transition was handled.

10. Practitioner review question

When an emergency or brownfield change makes live state disagree with declared intent, what evidence and authority should be required before that observed state is allowed to become the new desired state?

A counterexample that shows the transition model is too rigid, too weak or assigned at the wrong boundary is a useful result.

References

1. IETF, RFC 8342: Network Management Datastore Architecture, 2018.
2. IETF, RFC 9144: Comparison of NMDA Datastores, 2021.
3. NetBox Labs, Introduction to NetBox.
4. NetBox Labs, Custom validation.
5. NetBox Labs, Data protection rules.
6. NIST, SP 800-128: Security-Focused Configuration Management, 2011, updated 2019.
7. NIST, SP 800-53 Revision 5: Security and Privacy Controls, 2020.
8. OpenGitOps, GitOps principles.

9. Kubernetes, Controllers.
10. HybridOps Core, open-source implementation.
11. HybridOps documentation, Authoritative On-Prem Foundation showcase.
12. HybridOps Core, bootstrap NetBox operation.
13. HybridOps Core, authoritative foundation operation.
14. HybridOps Core issue #272, external technical review of Authority Before Automation v1.0.