

Was What's Running Approved?

Change control authority in Anaplan landscapes

Evidence of governed promotions, available on demand.

No drift between what was approved and what is running.

Nothing reaching production outside the path.

Most Anaplan landscapes can answer the question in the title with confidence. Far fewer can answer it with evidence. The distance between those two positions is small in ordinary operation and large during an audit, and closing it is the subject of this paper.

Valizant, Inc.

September 2026

Contents

01 **The control exists. It is executed by people.**

Every Anaplan change follows a process. It is also manual, and manual controls have known weaknesses and a measurable cost.

02 **A log is not a governance report**

What a governance report requires that a log does not, and why the record has to be assembled somewhere other than the platform it describes.

03 **The promotion mechanism is model-scoped**

What Application Lifecycle Management governs reliably, and what changes above it without moving through it.

04 **Three states**

Authorization attached to the change. Nothing reaching production outside the path. What was approved is what is running. Each stated as a condition a landscape can test.

05 **Where these functions have settled elsewhere**

For thirty years, conventional application architectures have pushed authorization, promotion, access restriction, and state comparison out of the application itself. Why that happened, and how much of it carries across.

06 **What changes**

The work is currently done twice, once at authorization and again months later for an auditor. The second time is avoidable.

07 **Was what's running approved?**

The question an auditor asks once a year, and the one a user asks when a page looks different from last week.

01. The control exists. It is executed by people.

Every Anaplan change follows a process. A request, an approval, a promotion scheduled and executed, and someone checking the result afterward. The process is real, it is documented, and in many organizations it works.

A promotion here is the move of a change from one environment to the next: development to test, test to production. Anaplan performs it through Application Lifecycle Management.

It is also manual, and manual controls have known weaknesses. There are labor costs for every instance. The process depends on the individuals who run it. It compresses under schedule pressure, because the release date is fixed and the checking is what gives. And it produces evidence only when someone sits down and assembles it, which happens after the fact, under time pressure, and from sources that were not designed to be assembled.

A manual control also looks only where someone has thought to look, so it misses things between audits. A page that drifted from the approved version, or a permission that widened during a release, is usually found by a user rather than by the control.

Not a criticism, just reality. This describes a control type every enterprise recognizes, and the usual response to a manual control is to automate it. That happened for moving code to production in traditional software, for financial close, and for certifying who has access to what. It has not happened for change to enterprise planning platforms, because the comparison between an approved state and a running one has to be made against a record the running system does not control, and that record has to sit somewhere else.

The cost is measurable. Practitioners running Anaplan landscapes at scale reported spending 100 to 400 hours per quarter on audit preparation, a range that scales with the number of models governed, the number of environments in the promotion path, and the number of systems the evidence is pulled from. Normalized to cost, these same practitioners put internal labor at \$100,000 to \$200,000 annually, with a further \$150,000 to \$300,000 one-time where a systems integrator designed the governance framework. Maintaining the scripts and tooling that support all of this consumed 20 to 30 percent of the platform team's engineering capacity, on an ongoing basis.

The spending does not buy better governance. The decisions were already made, the approvals already given, and the promotions performed correctly. The organization is paying, every quarter, to reconstruct a record of something it already did right.

02. A log is not a governance report

The reconstruction is necessary because of a distinction between two things that are routinely treated as the same.

A log answers one question: what changed. A governance report answers several, in sequence. Was this change authorized before it was made, and by someone with the authority to authorize it. Was it tested. Were the person who built it and the person who approved it different people.

Was it promoted through the governed process. And does what is running now match what was approved.

Only the first of those is a record of an event. The rest are relationships between events, and between events and artifacts that live somewhere else. The approval is in a ticketing system. The test result is in a spreadsheet or a screenshot. The promotion is in the platform. The verification is a sign-off in a release checklist, or a note closing the ticket in a third system. Each piece exists. What does not exist, until someone builds it, is the connective tissue that turns them into a single claim about a single change.

That assembly is the work priced in the previous section, and it is why better logging does not shorten it. A more complete log only answers the question a log can answer. It does not produce authorization, and authorization is what an auditor is testing.

There is a second reason the record has to be assembled, and it is specific to Anaplan. Model change history is retained by the platform as a permanent record, and it is reviewed in the interface. The Audit API, which is what an external system can read, carries recent activity. Both do what they were built to do.

What neither is built to do is hold evidence across an audit period. An audit conducted in March may examine changes made in the previous fiscal year, so whether that evidence can be produced depends on whether something outside the platform was retaining it at the time the change was made. That is a decision taken before the change, not after the auditor asks.

That is what an authority record does. One place that holds the authorization, the promotion, the comparison, and the result of each, for as long as the evidence has to last. Not because the platform's own records are deficient, but because a governance report is a relationship between records rather than any one of them, and something has to hold the relationship.

03. The promotion mechanism is model-scoped

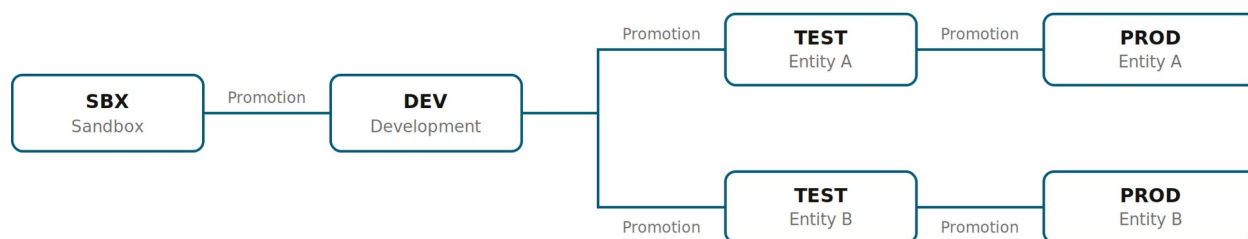
Anaplan provides a promotion mechanism for models. Application Lifecycle Management defines a source and a target, tracks structural revisions, and moves changes between them. Any organization running more than a single model uses it, and it does that job reliably: revisions are tracked, the comparison between source and target is exact, and the promotion either completes or does not. Its scope is the model, which follows from what a model is, a bounded versioned artifact with a defined structure that can be compared against another instance of itself. For moving model structure between environments, it is a solved problem.

Above the model sits a tier of capability that has grown considerably, and all of it is configured where it runs. Tenant administration, user and role assignment, security configuration. Workflow. Integration and orchestration. The application-level constructs that assemble models into the things users actually interact with. A change to any of it is made in the environment it affects, by a person with the rights to make it, and it takes effect there. What a promotion mechanism supplies elsewhere, a build step and a record of an approved state held outside the platform, is supplied by the process the organization puts around it.

Pages are the visible case. They are assembled in production and can be edited there, so a page can differ from the approved version without any model structure having changed, and users see the difference the moment they open it. Roles and permissions behave the same way: a role adjusted to resolve an access issue during a release alters what a group can see and do. Both are ordinary operational actions taken by people with the rights to take them. Both change what the business experiences, and neither appears in a record of what was promoted.

These changes are not invisible. Many are recorded. They are not governed by the same mechanism, so whatever authorization, testing, and verification exists around them is arranged separately, and in practice procedurally rather than structurally. A model change moves through a defined path. A tenant-tier change is made by a person who is trusted to follow the process.

That is a legitimate control. It is also a manual one, and it inherits every weakness described above.



The diagram above shows a promotion path of the kind common in a multi-entity landscape: a shared development environment feeding several test environments, each feeding its own production environment, with a version targeted across all of them. Each transition between environments is a promotion, and each promotion is a point at which authorization either exists and is attached, or exists somewhere else, or does not exist.

Two things follow from it. The first is how many such points an actual landscape contains. An organization running eight or twelve governed models across four environments is managing dozens of promotions per release cycle, each requiring its own authorization, testing evidence, and confirmation of result. That checking is executed once per promotion, not once per release.

The second is that the diagram shows only what moves through the path. It does not show what arrived by other means, because nothing in the promotion mechanism is looking for it.

04. Three states

The rest of this paper describes three properties a landscape either has or does not. Authorization attached to the change. Nothing reaching production outside the path. And what was approved being what is running. Each is stated as a condition rather than a capability, and each can be tested by asking a specific question and seeing whether the answer can be produced rather than asserted.

04.1 Authorization is attached to the change

The condition: for any change in production, the authorization for that change can be retrieved with it, as a single record, without reconstruction.

Most organizations have authorization. What they usually do not have is attachment. The approval exists in a change management system and references a ticket number. The change exists in the platform and references a model and a timestamp. A person who knows both systems can connect them. Nothing in either system does that.

Attachment means the authorization is a precondition of the promotion rather than a parallel record of it. The promotion does not proceed without it, and having proceeded, carries it. The test question is simple: pick a change from three quarters ago and ask how long it takes to produce the approval that authorized it, who approved it, and evidence that the approver was neither the person who requested the promotion nor the person who executed it. If the answer involves opening more than one system, the authorization is not attached.

There is a secondary property that matters more than it appears to. Attachment has to be arranged before the change, because that is when the authorization exists. Anything arranged afterward is reconstruction, however well organized.

04.2 Nothing reaches production outside the path

The condition: changes that did not move through the defined promotion path are detected, recorded, and dispositioned, rather than discovered incidentally.

A defined path is not the same as a path everything travels. Enterprise planning platforms permit changes to be made directly, by design, because operational reality requires it. An emergency fix at quarter end, a permission granted during an incident, a configuration adjusted to unblock a business user. They are the system working as intended under pressure.

The governance question is not whether they happen. It is whether the organization knows they happened. A landscape where direct change is possible and undetected cannot make a completeness claim, and completeness is what an auditor is testing when they ask how you know that all changes went through the process. Having a policy answers a different question.

Detection here means comparing what is present against what the governed process accounts for, at intervals short enough that the answer is still useful, and raising the difference for someone to explain. The output is not a list of violations. It is a record showing that the difference was identified, examined, and resolved, which is what a manual control cannot produce, because a manual control only looks where someone thinks to look.

What is being detected determines how much of a constraint the interval represents, and the cases differ. A revision tag and a promotion are durable artifacts. Once created they persist, and they surface whatever the interval, so a promotion cannot pass unseen between two comparisons by being undone. Detection of this kind reads the events the platform reports, and the events it reports are the ones that matter for a promotion.

Configuration at the tenant tier behaves differently, because not every change there produces an event to read. A page edited in place is a change of state rather than an event, and finding it

means comparing the state itself against the approved one, which is the third condition below. Where state is compared at intervals, a page or a permission that changed and then changed back between two comparisons leaves nothing to find. That window narrows as the interval shortens and does not close entirely.

One kind of change sits outside all of this by design. Model editing that does not receive a revision tag happens in a development environment, it is indistinguishable from ordinary modeling work, and it is not a change to a governed environment. It becomes one when it is tagged and moves with a promotion, which is the point at which the governed process has something to account for.

04.3 What was approved is what is running

The condition: the state of production has been confirmed to match the approved state, by something other than the process that performed the promotion.

This is the property organizations are least likely to have and least likely to notice they lack, because the promotion mechanism reports success and success is taken as confirmation. It is not. A successful promotion records that an operation completed. It is not an observation of the resulting state.

Most of the time the two coincide. Where they do not, the interesting cases are not the ones where something went wrong. They are the ones where nothing did.

A list's production data setting differs between two environments, so an authorized promotion preserves the target's data where it was expected to replace it. A modelling change resets the scope of a role or a selective access rule as a side effect, and people can see more or less than they could the day before. In each case the promotion was requested, approved, and executed correctly. Nobody acted on the user. Nobody would describe themselves as having changed access, or data, or anything but the model. The promotion record is clean and accurate, and it does not describe what happened to production.

Reference data behaves the same way and is worth naming separately. A list of currencies and conversion factors, or any similar table the model needs in order to compute, is not carried by a promotion. The new version requires it and it arrives by hand. That is a change to what production produces, made by someone with every right to make it, with no promotion record anywhere. It is dynamic data rather than structure, so comparing it is not in the current scope of either detection or verification.

The page and permission changes above are the more visible form of the same thing. None of these produces an error. All of them leave production in a state the approval did not describe, with a clean promotion record saying otherwise, and in most of them the people who notice first are the users.

Verification means the comparison is performed independently, on a schedule, against the approved baseline, and produces a record of its own. The test question is whether anyone has looked at production and compared it to the approval, rather than reading the promotion log. In

most landscapes the honest answer is that someone eyeballed the result on the day and has not looked since.

Most of what this returns arrives long before any auditor does. A page that differs from the approved version, a role granting more than it should, a promotion that reached three environments and not the fourth: today these surface when a user opens the page and something is missing, or is denied access at close, or reports a number that does not tie. Discovery runs through the business, and by the time it reaches the platform team, the disruption has happened, usually on the days the platform is relied on most.

A scheduled comparison moves discovery ahead of the user, onto the platform team's schedule, described the same way every time, in front of someone who can determine whether it was intended before anyone downstream is affected. That is an operational return rather than a compliance one, and it accrues every cycle whether or not anyone is asking for evidence.

It fits where the operating calendar already has natural points: after a promotion, on a routine schedule between releases, ahead of a period close, and before an audit. It asks nothing of the people who build models; it runs whether or not anyone remembers to run it, and each pass leaves a dated record of what was compared and what it found.

05. Where these functions have settled elsewhere

These are not exotic properties. Traditional software architectures have had versions of them for decades. Where do they normally live? In a conventional application architecture, the functions are distributed, and the distribution is consistent across organizations that otherwise share nothing. Source control holds the approved definition. A build pipeline moves it between environments. The authorization sits in a change management system. Direct edit of production is prevented by access control rather than by the application refusing it. Configuration management compares the running state against the definition on a schedule and reports the difference.

The pattern over three decades is that those functions stayed outside the applications whose changes they track. Source control, the change management system, and configuration management are external as a matter of course, and where a build pipeline has been offered inside a platform it has been offered alongside them rather than in place of them. That is not because application vendors were unwilling. Each function has a property that makes external residence necessary. The authorization has to be assertable independently of the system it describes. The comparison has to be made against a definition the running system does not control, since a system compared against its own account of itself cannot show a discrepancy. And the restriction on direct change is a question of who holds production access, which is an administrative control rather than a product feature.

The requirement is about where the record sits, not about who holds it. Change control and transport tooling often ship from the platform vendor itself, as separate products holding their own records, and that arrangement meets it. What cannot work is one system holding both the

approval and the state that approval is checked against, because then the comparison has no reference outside itself.

The achievement of conventional architectures was narrower than it is usually described. Drift did not go away. It went away for the artifacts that could be made immutable. A compiled binary or a container image has a hash, so the same artifact in two environments is identical or it is a different artifact. Source does not exist in production, only build output. Infrastructure stopped drifting once servers were replaced rather than patched, which removed the prior state instead of managing it. A schema version is answerable because a migrations ledger records what ran.

Where something stayed editable in place, the answer was comparison. Reference and seed data live in source control and are compared against the running database, because rows an application reads at runtime are part of its contract and do not otherwise stay aligned. Access is handled the same way: a declared policy and the permissions an identity actually holds are treated as two different things, and the second is computed rather than assumed, because a change to a role or a boundary moves it without anyone touching the identity. Environment configuration is compared against a declaration on a schedule.

So the pattern is not that some architectures solved this and others did not. It is that immutability solved it wherever it could apply, and comparison covered the rest. Data, access, and environment configuration are the categories where comparison remained necessary in architectures that had every other advantage.

A planning platform exists to be edited in place. A model is not compiled and shipped, it is built where it runs, and a page is assembled in the environment that serves it. That places more of the landscape in the category where comparison is the answer, and less in the category immutability covers.

Application Lifecycle Management is the exception, and a real one. It gives the model tier the property that immutability gives a build output, which is why moving model structure between environments is a solved problem. Above that tier, and in the cases inside an authorized promotion set out earlier, comparison is what remains. The response is the same response, applied to more of the surface.

Anaplan is not a conventional application architecture, and the analogy should not be pushed further than it goes. Models are not source code, model building is its own discipline with its own rigor, and the speed of change is the point of the platform rather than a risk to be managed. But the three properties above do not depend on any of that. They follow from what an authorization is and from what a comparison requires, and they hold wherever change is governed.

Salesforce is an instructive case on the remaining question, which is whether a native pipeline absorbs the layer above it once a vendor ships one. Salesforce shipped DevOps Center with promotion approvals and quality gates, natively and at no additional cost. The third-party market above it continued to grow. That is an observed outcome rather than a prediction, and it suggests the layer does not consolidate simply because the platform vendor enters it.

The Salesforce case also makes a further distinction visible. An approval gate operates at one moment, the promotion. It records that someone authorized that promotion, and once the

promotion completes it has done its work. Whether production still matches the approval a quarter later is a question about state rather than about authorization, and no point-in-time gate can answer it. A gate records a decision. A comparison observes a result. The distinction holds for a native gate, a third-party gate, and a manual one as well.

Two consequences follow. The first is about planning horizon. A change record, version history, and a promotion path are reasonable things to expect from a platform roadmap, and an approval gate may be, depending on whether the vendor treats promotion as a product surface. The comparison between approved and running state is a different matter, and thirty years of conventional software architectures suggest it does not arrive from inside the system being compared. A landscape that needs the comparison is better served by arranging it than by waiting for it.

The second is about position. What the three conditions above describe, taken together, is a change control authority: the function that holds the authorization, holds the approved state, and answers for the difference between that state and the running one. It sits above the promotion path rather than inside it. It does not move changes between environments and does not compete with anything that does. Its two records must live outside the system being governed, or they are the system reporting on itself.

06. What changes

Governance should require less effort than it currently does. Today the work is done twice: once when the change is authorized and promoted, and again months later when someone reconstructs the record of it for an auditor. The second time is avoidable, and the saving comes from moving the remaining effort rather than from doing less of it.

The reconstruction is where the 100 to 400 hours go, and it produces no governance that did not already exist. It produces the ability to demonstrate governance the organization needed at the moment of the change and could not capture then.

Capturing it at the moment instead means the authorization is recorded when it is given, the promotion carries it, the comparison runs on its own schedule, and the record accumulates as the work happens. The work at authorization is not zero, but it is work that was already being done. What no longer happens is the reconstruction.

One difference is what happens when a request arrives. Today, a request for evidence of a change made in the previous fiscal year initiates a search rather than a retrieval.

The other shows up sooner and is felt by more people. Where the comparison runs on a schedule, a page that drifted, a permission that widened, or a promotion that reached some environments and not others is identified by the comparison rather than by a user at close. The audit benefit arrives once a year. This one arrives every cycle.

Xentrl Assurance is built to produce that state on Anaplan. Three layers correspond to the three conditions above: Process Authority, which makes authorization a precondition of promotion and attaches it; Out-of-Process Detection, which identifies changes that did not move through

the process; and State Verification, which confirms that production matches the approved state. The first two are available now. State Verification is expected in 2027.

It operates alongside the platform's own mechanisms rather than replacing them. Where a promotion path exists, it governs that path. Where changes are made directly, it detects and records them. The platform continues to do what it does, and the authority record sits outside it, which is where a record that answers for the platform's state has to be.

07. Was what's running approved?

This question is answerable in every landscape. The organization knows what it approved, and the people involved remember doing it correctly, and in most cases they are right.

What varies is whether the answer can be produced or only asserted. Producing it requires that the authorization was attached when it was given, that anything arriving outside the path was noticed at the time, and that someone has compared the running state to the approved one since. All three are decisions taken before the question is asked. None of them can be arranged afterward, which is why the reconstruction is expensive and why it recurs every quarter.

Of the three, the comparison is the one most often absent and the least demanding to put in place. It requires no change to how models are built, no additional step for the people making changes, and no reorganization of the release process. It also pays before it is asked to. A difference between the approved state and the running one reaches users whether or not an audit is pending, and a scheduled comparison is what finds it first. It runs on a schedule against the approved baseline and accumulates its own history, so that the answer to the question in the title is not assembled when it is asked. It is already there, dated, and covering the period the question is about.

The question in the title is the one an auditor asks once a year. It is also, in less formal words, what a user is asking when a page looks different from the one they used last week. A landscape where the three conditions are met answers both without assembling anything.