CLOUDBOLT

COMPLEMENTARY STACK SERIES: PART 1

Same Automation Market, Different Jobs: Why Ansible, Terraform, and CMP Aren't Actually Competing

A prospect recently told us that a Red Hat rep had described Ansible Automation Platform as a replacement for the kind of self-service catalog a cloud management platform provides. It is not an isolated pitch. As platform engineering has become the default operating model for IT, Gartner expects 80% of large software organizations to have dedicated platform teams by 2026, nearly every vendor in the automation space has added a self-service front end to its product and started describing it as a platform in its own right. Ansible now ships a self-service automation portal. Terraform has no-code modules with a form-based UI. Backstage built an entire ecosystem around service catalogs. None of that is dishonest marketing, exactly, but it does create real confusion for the people who have to defend a budget line or answer a competitor's claim with something more solid than a hunch.

FOUR LAYERS, ONE STACK

The clearest way through the confusion is to separate what each category of tool is actually built to do, not what its newest feature slide claims.

Configuration management ANSIBLE	Configuration management, the category Ansible built its reputation on, executes tasks against systems that already exist: installing packages, applying settings, enforcing state. Red Hat's own language for Ansible Automation Platform's newest positioning calls it the trusted execution layer for IT operations, which is a precise and accurate description. It runs the work, reliably, at scale.
Infrastructure as code TERRAFORM	Infrastructure as code, Terraform's category, provisions the resources themselves: the servers, networks, and cloud services a workload needs to exist in the first place. HCP Terraform's no-code modules let a platform team publish an approved module behind a form, so a requester can provision that one specific thing without learning HashiCorp Configuration Language, but the workflow starts and ends inside Terraform's own state and its own resources.
Internal developer platforms BACKSTAGE	Internal developer platforms, the category Backstage popularized, are catalogs and portals: a place to see what services exist, who owns them, and how to scaffold a new one from an approved template. Backstage itself is explicit that it is portal only, with no execution layer of its own. Every action it triggers has to be wired to a plugin that calls something else.
Cloud management platform CLOUDBOLT CMP	A cloud management platform is the fourth layer, and it is a different job entirely: coordinating the approvals, governance, cost controls, and workflow logic that sit around and across all of the above, regardless of which tool actually executes the work.

READING THE STACK

Each tool below is doing a different job, in a different part of the request lifecycle: Ansible executes, Terraform provisions, Backstage catalogs. None of the three approves anything, which is why the approve stage sits empty in every row. That gap, not the execution work, is what a cloud management platform is for, and it is why CloudBolt runs across all five stages instead of owning one.

STOP COMPARING DIFFERENT LAYERS OF THE STACK

COMMON MISCONCEPTION: These tools compete with each other.

REALITY: They solve different problems at different layers of the platform engineering stack.

CONFIGURATION MANAGEMENT (Ansible)

PURPOSE

Execute tasks on existing systems: configure, patch, enforce state.

NOT BUILT FOR

- Provisioning infrastructure
- Catalogs & service discovery
- Governance & approvals

INFRASTRUCTURE AS CODE (Terraform)

PURPOSE

Provision the infrastructure resources workloads need.

NOT BUILT FOR

- Approvals & policy enforcement
- Cost governance
- Catalogs & portals
- Executing configuration

INTERNAL DEVELOPER PLATFORM (Backstage)

PURPOSE

Discover services and scaffold new ones from approved templates.

NOT BUILT FOR

- Governance & approvals
- Provisioning infrastructure
- Executing work

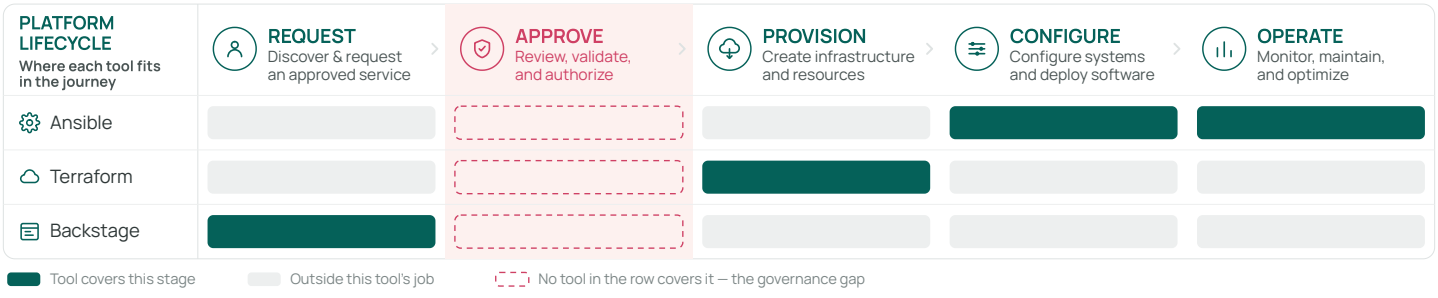
CLOUD MANAGEMENT PLATFORM (CloudBolt)

PURPOSE

Coordinate, govern, and orchestrate all platform operations.

- Governance & policy
- Workflow & approvals
- Cost management
- Orchestration across tools
- Self-service enablement
- Integrations & extensibility
- Full lifecycle visibility

How each tool fits in the platform lifecycle



CloudBolt (Governs Across All Stages)

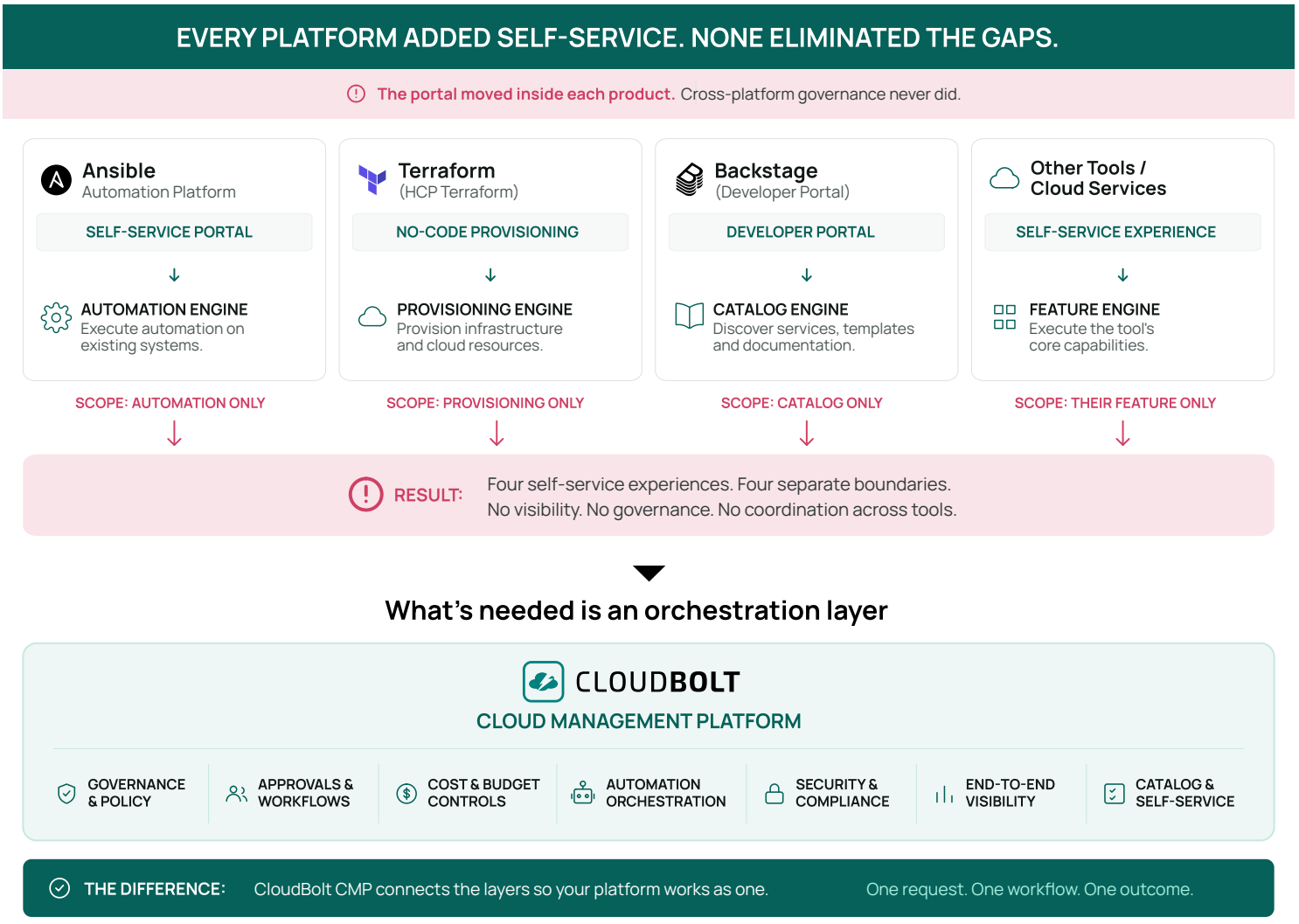
- Policy & Guardrails Enforced
- Approvals & Workflows
- Cost & Budget Controls
- End-to-End Visibility & Audit Trail
- Risk, Compliance & Reporting

THE BOTTOM LINE: Different tools. Different jobs. One cohesive platform experience.

CloudBolt connects the layers so your platform works as one.

WHY THE LINES ARE BLURRING RIGHT NOW

None of these vendors are confused about their own architecture. What has changed is the market around them. Platform engineering has made self-service the feature every buyer wants to hear, so every vendor has built a version of it scoped to its own category. Ansible Automation Platform 2.7 added a self-service automation portal with a centralized content catalog, aimed at teams who want to request Ansible-run automation without opening a playbook. HCP Terraform's no-code provisioning does the same thing for Terraform modules. Both are genuinely useful. Neither one governs what happens outside its own tool: Ansible's portal does not manage Terraform's state, and Terraform's no-code catalog does not approve or track what a configuration management job does on the resulting server. The self-service layer got added inside each category. The gap between categories did not close, it just got harder to see on a slide.



THE DEFENSIBLE VERSION FOR A BUDGET CONVERSATION

When a vendor pitches its tool as a replacement for a governance and orchestration layer, the useful question is not whether that tool has a catalog. Most of them do now. The useful question is whether that catalog can approve a request that spans Ansible and Terraform and a public cloud API in one workflow, track cost and ownership across all of it, and keep working the same way if the execution tool underneath ever changes. That is a different job than executing a playbook or provisioning a module well, and it is the job a cloud management platform is built for specifically.

This is also why CloudBolt does not treat Terraform, Ansible, Puppet, or Chef as competitors: they are listed among its 200-plus integrations, because CMP is designed to orchestrate them, not replace them. Provisioning creates a resource. Orchestration is what guarantees that resource was requested correctly, approved by the right people, tagged and costed properly, and built the same way the next hundred times someone asks for it. A team does not have to choose between Ansible and a governance layer, or Terraform and a governance layer, any more than they would choose between a database and a backup tool. The two solve different problems, and the organizations with the clearest budget conversations are the ones that can say exactly which problem each tool in their stack is solving.

THE BOTTOM LINE

Ansible, Terraform, and Backstage are not competing with a cloud management platform, no matter how their newest self-service feature gets pitched.

Each one does its own job well: executing automation, provisioning infrastructure, and cataloging services. None of them coordinate approvals, governance, and cost across all three at once, and across whatever platform is underneath. That is the layer this series looks at next, tool by tool, starting with the one causing the most confusion in sales conversations right now: Ansible Automation Platform.

Talk to CloudBolt cloudbolt.io/demo