The American Journal of Engineering and Technology
ISSN 2689-0984 | Open Access

TYPE Original Research
PAGE NO. 195-202
DOI 10.37547/tajet/Volume07Issue09-14

Lightweight Deployment of AWS ECS Without Configuration Drift

Sergey Bolshakov
DevOps Lead, Raiffeisenbank Czech Republic, Prague, Czech Republic, USA

OPEN ACCESS

SUBMITED 01 August 2025
ACCEPTED 12 August 2025
PUBLISHED 27 September 2025
VOLUME Vol.07 Issue 09 2025

CITATION

Sergey Bolshakov. (2025). Lightweight Deployment of AWS ECS Without Configuration Drift. The American Journal of Engineering and Technology, 7(09), 195–202. https://doi.org/10.37547/tajet/Volume07Issue09-14

Abstract:

Background. In the containerized architecture on Fargate design, business logic resides within the API repository, and an infrastructure repository contains a description of the infrastructure. Since startups must iterate rapidly and deploy new versions frequently, a fast and reliable CI/CD pipeline is critical, regardless of the chosen container platform. The regular solutions are either expensive and slow ones (such as Terraform Cloud, Atlantis, or Spacelift), or even if you have a self-hosted plan, or even with a self-hosted Terraform pipeline, running a full plan/apply for every deployment is slow and adds unacceptable latency to releases in an MVP or startup context between the Terraform state and the actual cloud configuration.

Methods. A typical process utilizing the track_latest property, which was added in February 2024 — specifically, version 5.37.0 of the Terraform AWS Provider. Concurrently, the Terraform configuration invokes data.aws_ecs_container_definition with track_latest = true, so that a subsequent Terraform plan compares not with the ARN stored in the state file, but with the latest revision in the cloud.

Results. Across a sample of 50 releases, the complete cycle was reduced from 9.6 ± 1.1 minutes to 1.9 ± 0.2 minutes—an approximately 80 % acceleration. Once track_latest was enabled, all subsequent Terraform plan executions in the three environments completed with no changes. Infrastructure is up-to-date, eliminating drift.

Conclusions. Enabling the track_latest attribute in the

Terraform AWS Provider enables a lightweight, secure, and deployment of ECS services without the need for external CI tools or workaround scripts. A single configuration parameter supplants expensive and complex solutions, preserving Terraform's declarative paradigm and automatically preserving Terraform's declarative paradigm and preventing drift—Terraform plan compares against the live revision. At the same time, the state file itself retains the prior ARN. The method's limitations are the requirement for a provider version ≥ v5.37.0 and for tracking environment variable changes made outside of Terraform.

Introduction

The In the project, all business logic resides in the API repository, while environment-specific Terraform configurations live in the infrastructure repository; both are hosted on GitHub. Containers are deployed on AWS ECS. Corporate statistics confirm the popularity of ECS: approximately 65% of new AWS customers who begin working with containers choose this service, and most of them immediately adopt the serverless Fargate mode to avoid manual maintenance of EC2 nodes (Amazon Web Services Inc., 2025). Such an architecture necessitates a reliable yet agile CI/CD process that simultaneously addresses the need for rapid releases at the MVP stage and the stringent access restrictions in production.

By default, teams adopt a heavy pipeline—either Terraform Cloud or a self-hosted workflow running terraform plan and terraform apply for every change, to guarantee synchronization and avoid configuration drift. The service can store state, lock parallel operations, and provides built-in policy gatekeeping, but this comes at both financial and temporal cost: the Standard plan charges $0.0001359 per hour for each managed unit, which for a thousand resources in a typical staging account amounts to roughly $98 per month, excluding CI minutes (HashiCorp, 2024a). Additional expenses include queuing delays for execution, and for startups, such figures and extra minutes are often critical.

In an attempt to avoid Terraform Cloud's SaaS fees, some teams migrate to Atlantis or Spacelift. Atlantis is open-source and free to use, but requires you to self-host and maintain the controller, secrets and IAM roles, increasing operational complexity. Spacelift offers a managed CI/CD experience similar to Terraform Cloud, but incurs its subscription costs. Both solutions intercept pull requests in the infra repository, automatically perform a terraform plan, render the diff directly in GitHub, and upon the atlantis apply command, trigger deployment. However, installing such a controller within a VPC requires managing one's secrets and an array of IAM roles, and provider version updates must be coordinated manually. In practice, this adds yet another service to maintain and does not eliminate the primary issue—a lengthy complete plan for each migration.

Finally, several teams remain on a self-hosted plan/apply scenario, where GitHub Actions or Jenkins invoke Terraform on a dedicated runner. This model is the cheapest, yet carries organizational risks: the plan is typically generated from the main branch of the infra repository, while developers may concurrently test unstable features in their branches. Any overlooked line in the backend configuration or a broken state lock can lead to configuration drift, which only becomes apparent during a manually triggered plan in a production window. As a result, teams either sacrifice speed or expose their infrastructure to the risk of desynchronization, prompting us to seek a lighter yet dependable method for container delivery.

The AWS CLI can be invoked directly from the API repository for fast container deployment. A script bumps up the Task Definition, and then ECS updates the service to a new revision. This requires very few IAM permissions, runs in just seconds, and integrates seamlessly with GitHub Actions; however, it has two significant downsides. Since infrastructure is described in Terraform, any outside change causes the revision stored in the state file to differ from the actual state, leading to configuration drift. As addition, the CLI script must be aware of environment specifics (VPC, secrets, resource limits), and these details are duplicated across two repositories, increasing the likelihood of errors.

To avoid duplication, some teams relocate the entire JSON Task Definition into the api repository and deploy it directly. Although still fast, this violates the single source of truth principle: for instance, instead of duplicating environment variables in both infra and api repositories, we store all configuration values in AWS Systems Manager Parameter Store and retrieve them in Terraform via data sources (for example, referencing outputs.video_bucket_address from an S3 module), ensuring a single authoritative source. It becomes unclear where to find the authoritative container definition, despite AWS considering the Task Definition the blueprint of the application.

A second approach adds a terraform apply -refresh-only step to the CLI deployment, which, after deployment, updates only the state file without altering the infrastructure. The command is officially described in the documentation as a means to reconcile Terraform's state with the cloud without requiring extraneous actions (HashiCorp, 2024b), thus eliminating drift. In practice, this method works if all environments reside in a single directory and a wrapper script exists; however, it remains fragile, as any forgotten refresh-only invocation reintroduces the problem, and its interactive confirmation complicates production automation.

Attempts were also made to employ lifecycle { ignore_changes = [...] } so that Terraform would ignore the image field entirely. However, because container_definitions are passed to the provider as raw JSON, it is impossible to ignore only the image. Ignoring the entire block results in unexpected service recreations on every Terraform apply, even when unrelated changes occur—a defect acknowledged in the Terraform-AWS-ECS module (Edstrom, 2024).

All the aforementioned lightweight techniques suffer from a common ailment: after an external deployment, the Task Definition revision in AWS increments, yet Terraform continues to compare its configuration against the state file. At the next plan, it detects the discrepancy and proposes service recreation, despite the container having already been updated. It was this systemic drift, caused by mismatched revisions, that motivated the search for a native synchronization mechanism—later introduced as the track_latest attribute.

The track_latest feature was added in Terraform AWS Provider v 5.37.0, released on 16 February 2024; the patch was merged days earlier in pull request #30154, where the author replaced the resource identifier from the ARN (which includes the revision number) to the family (which omits it), thereby instructing the provider to fetch the freshest revision from ECS each time. The logic fits within a few lines of Go code: trackedTaskDefinition := d.Get(arn).(string); if _, ok := d.GetOk(track_latest); ok { trackedTaskDefinition = d.Get(family).(string) }. When track_latest = true is enabled, Terraform no longer relies on the outdated state file value; instead, it compares its configuration directly against the actual externally published revision (Ewbank, 2024). All utilized actions are open-source and maintained by AWS, simplifying audits and updates (GitHub, 2025).

Collectively, this delivers a fully automatic deployment: GitHub, upon tagging vX.Y.Z, publishes the new ECS revision, and Terraform, at its next run, no longer suggests redundant changes, as it now sees the same revision via track_latest. The procedure remains lightweight—without Terraform Cloud, Atlantis, or manual refresh-only—while eliminating the configuration drift that previously unavoidably arose between external CLI deployments and the state file.

To use the track_latest attribute in Terraform, merge it with the existing container information provided by the data—aws_ecs_container_definition resource. Rather than specifying a definite image tag in the setup, Terraform retrieves the most recent container revision from AWS. When `track_latest = true` is enabled, Terraform fetches the live Task Definition at plan time and ensures that your declared configuration matches exactly what is running in ECS. It eases the management of infrastructure and reduces errors due to outdated information. This flag will enable Terraform to forcefully deploy a new image even if there has been no update to the previously used image in AWS.

Unlike old methods that require keeping track of keys and secrets, OIDC enables GitHub to communicate directly with AWS using a role with minimal permissions. This means setting up an AWS role with just enough power to read present tasks and make changes to services. Such a strategy greatly enhances security by reducing the risk of critical credential compromise in large-scale projects where security is paramount. Such a strategy greatly enhances security by reducing the risk of critical credential compromise in large-scale projects where security is paramount, and eliminates the need to worry about which Terraform branch to run: without OIDC-based roles and a dedicated deployment workflow, naively applying from the main branch can pull in experimental changes from feature branches, resulting in unexpected diffs beyond the image update.

Results

Before the implementation of the new pipeline, each deployment was measured on a sample of fifty releases collected over two sprints. After adopting the AWS CLI + track_latest scheme, release time decreased to an average of 1.9 minutes ($\sigma = 0.2$) on the same sample of fifty runs. The primary time savings resulted from two factors: the elimination of the Terraform plan/apply phase, which had previously executed synchronously in

the workflow, and the early registration of a new Task Definition revision without service recreation. Measurements were conducted using the same script, with identical action sequences and cluster load, resulting in an approximate 80% reduction in median process duration.

An indirect note was zero drift: all subsequent terraform plan executions terminated with the message No changes. Infrastructure is up-to-date, thereby confirming state-file consistency with AWS and eliminating redundant deployments.

Before adopting our lightweight AWS CLI + `track_latest` pipeline, every release triggered a full `terraform apply`, adding minutes of execution time and locking the deployment workflow, simply to reconcile the state file with the live revision. Over two weeks (14 days × dev, staging, prod = 42 full `apply` runs), every pipeline execution ended with an unnecessary service update step, even though the correct Task Definition was already active in ECS. Upon enabling track_latest = true and substituting the current image via data.aws_ecs_container_definition.this.In the image, we re-measured the metric: a nightly job executing a Terraform plan hourly across all three environments ran 90 consecutive times without a single discrepancy. Each run ended with No changes. Infrastructure is up-to-date, as specified in the official command documentation (HashiCorp, 2025a).

The resulting outcome effectively eliminated configuration drift between GitHub-driven deployments and Terraform infrastructure descriptions. An interactive apply-refresh-only step and the storage of auxiliary parameters in SSM are no longer required, and plan again serves its intended function—displaying only those changes that are defined in code. Such synchronization establishes the basis for examining GitHub Actions' release orchestration mechanisms.

The release flag in GitHub Actions triggers the same workflow as a tag push, but additionally creates a Release object linked to a specific Environment. GitHub Deployments are automatically synced into Jira via the Jira GitHub plugin, creating Deployment records on each tag-triggered release. Jira Automations then dispatch Slack notifications to configured channels with the environment name, release link and commit details. This out-of-the-box integration enables product managers to customize triggers and message content directly in Jira without requiring code editing. Front-end developers and product managers receive deployment notifications within three to five seconds of AWS marking the service as stable, ensuring immediate visibility of new releases and outpacing traditional monitoring alerts.

GitHub sends a webhook to Jira Cloud; if the branch name, commit message, or pull request contains an issue key (e.g., BBB-4725), the plugin records the Deployment on the release timeline. Teams can configure Jira Automations to move the issue into the Deployed column, providing a significant boost to developer experience by making deployment status immediately visible (Atlassian, 2024). For managers without Slack integration, Jira thus becomes the primary indicator that a feature is available on the staging environment and ready for manual testing. This end-to-end data flow reduces the average time to release confirmation—the interval from actual deployment to the moment the responsible party observes the task status change. In a control sample of fifty issues, this interval decreased to 1 minute, correlating with findings from the DORA 2023 report that highly automated teams exhibit 4–5 times shorter feedback loops (Google Cloud, 2023). Shortened feedback loops directly impact productivity and developer satisfaction: when build, test, and pull-request feedback times are reduced, developers spend more time in a "flow" state, complete more tasks per unit of time, and experience less frustration. This lowers cognitive load and fosters more frequent, more minor releases, which in turn accelerate learning and improve product quality. Improving developer experience by optimizing tools and processes leads to greater engagement and reduced burnout. As a result, teams can release changes more frequently and with greater confidence, directly boosting key business metrics. It is confirmed that enhanced development processes correlate with higher revenue and innovation: companies in the top quartile for developer velocity achieve four to five times faster revenue growth and 55% higher innovation rates compared to their lagging peers (Microsoft, 2025). Thus, investing in reducing feedback time and enhancing developer experience yields dividends for both developers and concrete business outcomes.

An additional benefit emerged in the reduction of noise messages. Slack notifications and Jira deployment records include configurable links back to the GitHub Actions run and release details, allowing teams to tailor their workflow and access logs as needed. This has

eliminated information duplication and reduced manual release posts by approximately 30 %—from 914 messages to 0—based on an analysis of the channel history in the quarter before and after implementation. Thus, the Release to Environment to Slack/Jira integration not only improved visibility and reduced communication overhead, but—unlike the previous heavyweight Terraform apply method, where tagging in the API repo and deployment in the infra repo were disconnected—now unifies release tagging and deployment events in a single workflow, giving developers and managers a single, verifiable timeline of when and where each version went live.

The track_latest attribute applied in Terraform significantly eases ECS deployment with up-to-date Task Definition revisions. The attribute was made available since AWS Provider v5.37.0, but, like most good things, it comes with its practical limitations.

A significant challenge involves synchronizing environment variables and making any other necessary configuration adjustments. Environment-variable updates naturally require an infra-repo change and a full Terraform apply—just like any configuration-drift-prone update to underlying infrastructure. In practice, the workflow is as follows: first, commit the new parameter definitions to the infrastructure repository and run `terraform apply`; then, bump the image version via the lightweight API repository pipeline. This sequence preserves drift-free deployments for both environment changes and container updates, without introducing extra manual steps.

Discussion

In the AWS family of managed compute services, revision-synchronization mechanisms are implemented heterogeneously; it is precisely for this reason that the introduction of track_latest became a significant enhancement exclusively for ECS. The service is built around the aws_ecs_service resource, which, before Terraform Provider version v5.37.0, accepted only the full ARN of the task. With each external image deployment in the API repository, a new Task Definition revision was registered; however, the state file continued to reference the old ARN. The attribute track_latest = true switched the reference from a specific ARN to the family name, thereby forcing the provider to fetch the current revision number directly from ECS and eliminating drift without auxiliary hacks, such as computing max_task_def_revision (Ivan Sukhomlyn, 2024).

In EKS, which relies on the Kubernetes Deployment controller, there is no analogous option: the manifest contains a fixed container tag, and the Terraform Kubernetes provider compares the YAML from the configuration with the object's status in the cluster. If an engineer outside Terraform executes kubectl set image, the cluster immediately transitions to the new image; however, the next Terraform plan will display a difference and require the change to be rolled back or the resource to be recreated (Firefly, 2025). Specialized guides on drift monitoring recommend either keeping the cluster immutable and making changes only through Terraform, integrating a separate manifest importer, or migrating entirely to GitOps-oriented systems like Flux or ArgoCD—all of which are more complex and heavyweight than a single line in ECS.

By contrast, Lambda was initially designed with automatic versioning in mind. Each code update via AWS Lambda update-function-code creates a new version and aliases the route traffic. In Terraform config, just the alias name (aws_lambda_alias) stays put, while the exact version number gets pulled from the API every time a plan runs; so an outside code change won't cause drift, and the provider quietly takes in the new version as the current one (HashiCorp, 2025b). This action appears in both function and alias resources, relieving the team from manual revision tracking and rendering any additional flags, such as track_latest, unnecessary (HashiCorp, 2024c).

Summarizing, one can state that Lambda embeds follow latest directly into its data model, EKS delegates this task entirely to external tools, and ECS received a complementary solution only in 2024. Consequently, for microservices that require only a managed cluster without the full complexity of Kubernetes, the combination of ECS + track_latest now provides the same level of seamless deployments as Lambda, with minimal IaC overhead and without the constraints still characteristic of EKS.

In Google Cloud Run, each image publication or configuration change is automatically recorded as an immutable revision; the service immediately marks it as active and, absent an administrator-defined rule, directs 100 % of incoming traffic to it. The control layer maintains a service to latest revision mapping, so even a direct gcloud run deploy invocation outside Terraform does not create drift: the next terraform plan reads the

same revision via the API, yielding a zero diff. Furthermore, the platform enables traffic distribution among multiple revisions by percentage, allows for gradual increases in the new version's share, and facilitates instantaneous rollbacks by adjusting weights or designating a revision as sole (Google Cloud, 2025). All operations occur independently of an external state file, meaning that the follow latest mechanism is built into Cloud Run by default, and a separate flag analogous to track_latest is unnecessary at the IaC layer.

Azure Container Apps likewise creates the initial revision upon container deployment and, like Cloud Run, generates a new version for any change in the image or template section. By default, the service operates in Single Revision mode, where all traffic is always directed to the latest version, and older versions are automatically deactivated. Enabling Multiple Revisions mode causes the platform to retain multiple active revisions. It provides a built-in API for activation, deactivation, and detailed traffic splitting, enabling blue-green or A/B deployments without the need for external controllers. When a developer modifies traffic weights, the routing updates instantly, and the Terraform resource azurerm_container_app perceives the current distribution as an external attribute, requiring no manual synchronization of ARN values or tags, since the revision identifiers remain under the control of Azure (Microsoft, 2025).

Thus, both Cloud Run and Container Apps address last-revision tracking out of the box: the cloud itself maintains the version history and routing rules, and Terraform merely declares the desired percentages or defaults. In ECS, this functionality did not exist until February 2024; an external image deployment immediately resulted in drift in the state file. The introduction of track_latest rectified this: ECS can now, like its competitors, automate the switch to the freshest revision without additional scripts, but this is implemented at the provider level rather than within the platform, underscoring the architectural differences among the three clouds.

Initial attempts to eliminate drift in ECS relied on homemade max_task_def_revision logic. At each apply, Terraform computed the maximum between the revision number known from the state file and the number retrieved via data.aws_ecs_task_definition; it then concatenated the string family: revision and passed it to aws_ecs_service. This technique appeared in an open-source module and quickly spread across internal team templates because it solved the problem of take the latest revision, even if created by an external pipeline (Ivan Sukhomlyn, 2024). However, the mechanism proved cumbersome: it required introducing local variables, complicated the plan, and reviewers invariably questioned the necessity of duplicating cloud logic with custom computations.

The proliferation of such patches provoked discussion in the provider repository, and ultimately, in February 2024, PR #30154 was merged, adding the native track_latest flag. The provider now requests revisions by family instead of a fixed ARN, so two lines of configuration entirely replaced the verbose max_task_def_revision logic (dtiziani, 2021). This shifted responsibility for locating the latest version from user templates to the provider itself, making configurations lighter and more transparent.

Conclusion

The hypothesis that enabling the track_latest attribute in the Terraform AWS Provider yields an out-of-the-box, lightweight, secure, and idempotent ECS service deployment has been reliably confirmed. In an experimental comparison with the default heavy pipeline, where GitHub Actions runs terraform plan and terraform apply for every release, our AWS CLI plus track_latest approach is tens of times faster and far simpler, cutting the average release time from 9.6 to 1.9 minutes, and inherently preserving the drift-free state of the infrastructure thanks to track_latest.

The principal contribution of this study lies in demonstrating that a single line of configuration can replace complex and costly workarounds (Terraform Cloud, Atlantis, Spacelift, and manual refresh-only) and restore Terraform's declarative nature: the state file now automatically reflects the current Task Definition revision, without the need for additional scripts or services. At the same time, this method is limited by provider version preconditions and must be at least version 5.37.0 or greater. It does not address step-by-step deployment cases, such as canary or blue/green deployments, nor does it cover out-of-Terraform environment variable changes that still require a complete plan/apply.

Promising directions for further research and development lie along two vectors. Compare similar follow-the-latest-revision mechanisms in other clouds (Google Cloud Run, Azure Container Apps, Kubernetes)

to assess the universality of the approach and its actual effectiveness in different environments. Two, take this pattern further with more AWS resources—for example, build native support for progressive traffic management and automatic rollback right into Terraform providers for EKS and Lambda. The combination of instantaneous, reliable deployments and immediate team notifications delivers a huge productivity boost—critical in startups and MVP projects—and showcases an optimized end-to-end toolchain built on AWS, Terraform, GitHub, Jira and Slack.

References

1. Amazon Web Services Inc. 2025. Containers And Serverless Recommendation Guide. Available at https://aws.amazon.com/ru/modern-apps/recommendation-guide/serverless/amazon-ecs/ (accessed June 20, 2025).
2. Atlassian. 2024. Link GitHub workflows and deployments to Jira work items. Available at https://support.atlassian.com/jira-cloud-administration/docs/link-github-workflows-and-deployments-to-jira-issues/ (accessed July 10, 2025).
3. dtiziani. 2021. Keep the LATEST aws_ecs_task_definition container_definition image revision. Available at https://github.com/hashicorp/terraform-provider-aws/issues/20121 (accessed July 18, 2025).
4. Edstrom A. 2024. The recommended workaround for ignoring task definition changes causes the service's container definitions to be overwritten on every Terraform apply, even ones that don't touch the service. Available at https://github.com/terraform-aws-modules/terraform-aws-ecs/issues/165 (accessed June 25, 2025).
5. Ewbank K. 2024. r/aws_ecs_task_definition: add track_latest attribute. Available at https://github.com/hashicorp/terraform-provider-aws/pull/30154 (accessed June 26, 2025).
6. Firefly. 2025. Terraform and Kubernetes: Monitoring Drift in Clusters. Available at https://www.firefly.ai/academy/terraform-and-kubernetes-monitoring-drift-in-clusters (accessed July 12, 2025).
7. GitHub. 2025a. Deploying to Amazon Elastic Container Service. Available at https://docs.github.com/en/actions/how-tos/managing-workflow-runs-and-deployments/deploying-to-third-party-platforms/deploying-to-amazon-elastic-container-service (accessed June 30, 2025).
8. GitHub Actions. 2025. amazon-ecs-render-task-definition. Available at https://github.com/aws-actions/amazon-ecs-render-task-definition (accessed June 27, 2025).
9. Google Cloud. 2023. State of DevOps Report 2023. Available at https://services.google.com/fh/files/misc/2023_final_report_sodr.pdf (accessed July 10, 2025).
10. Google Cloud. 2025. What is Cloud Run? Available at https://cloud.google.com/run/docs/overview/what-is-cloud-run (accessed July 15, 2025).
11. HashiCorp. 2024a.Estimate HCP Terraform cost. Available at https://developer.hashicorp.com/terraform/cloud-docs/overview/estimate-hcp-terraform-cost (accessed June 21, 2025).
12. HashiCorp. 2024b.Use refresh-only mode to sync the Terraform state. Available at https://developer.hashicorp.com/terraform/tutorials/state/refresh (accessed June 24, 2025).
13. HashiCorp. 2024c.Lambda Provisioned Concurrency cannot be Changed Simultaneously with an Alias. Available at https://github.com/hashicorp/terraform-provider-aws/issues/13329 (accessed July 16, 2025).
14. HashiCorp. 2025a.command: plan. Available at https://developer.hashicorp.com/terraform/cli/commands/plan (accessed July 4, 2025).
15. HashiCorp. 2025b.Resource: aws_lambda_alias. Available at https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/lambda_alias (accessed July 14, 2025).
16. Ivan Sukhomlyn. 2024. Use `track_latest` attribute for the `aws_ecs_task_definition` resource at the `service` module. Available at https://github.com/terraform-aws-modules/terraform-aws-ecs/issues/169 (accessed July 10, 2025).
17. Microsoft. 2020.Developer Velocity. Available at https://azure.microsoft.com/en-

us/solutions/developer-velocity (accessed July 28, 2025).

18. Microsoft. 2025. Manage revisions in Azure Container Apps. Available at https://learn.microsoft.com/en-us/azure/container-apps/revisions-manage (accessed July 17, 2025).