

International Journal of Computing and Engineering

(IJCE)

**DriftAgent: Using a Language Model to Turn Terraform Drift
into Reviewed Merge Requests**



CARI
Journals

DriftAgent: Using a Language Model to Turn Terraform Drift into Reviewed Merge Requests

 Nitish Mane

Infrastructure Engineering, Philo, Inc., San Francisco, California, USA

<https://orcid.org/0009-0000-0488-2897>

Accepted: 12th Sep, 2026, Received in Revised Form: 28th Sep, 2026, Published: 12th Oct, 2026

Abstract

Purpose: This paper describes the design and operation of DriftAgent, a system that detects configuration drift in Terraform-managed cloud infrastructure and prepares fixes for human review. It also reports what we learned from running the system in production at a streaming television company.

Methodology: The study used a single-site case study design to evaluate a production deployment. DriftAgent runs every weekday morning: it asks an Atlantis server to plan 187 Terraform projects in parallel, parses the plan output, and sends each drifted project, together with its source files, to a large language model in a single request. The model returns HCL changes, including Terraform import blocks for resources that exist in the cloud but not in state; the agent opens a GitLab merge request for human review and posts a daily summary to Slack. Over twelve weeks (56 runs), we measured four outcomes: the fate of each merge request (merged unchanged, merged after edits, or closed), plan sweep time, engineering time per drift event, and inference cost. Merge request outcomes were validated by two engineers who independently rated a random sample of 120 merge requests as acceptable, acceptable with edits, or unacceptable against defined criteria, and inter-rater reliability was measured with Cohen's kappa. Engineering effort was compared with a pre-deployment timing study of 71 manually handled drift events. Data were analyzed with descriptive statistics (counts, percentages, means, medians, and 95th percentiles), edited or closed proposals were classified by their main failure reason, and the findings were synthesized by triangulating production logs, merge request histories, the independent ratings, and the timing data.

Findings: Across 56 runs the system recorded 372 drift events and opened 351 merge requests. Reviewers merged 73.5% of them without semantic changes and 18.2% after edits, and closed 8.3%. Moving plan execution from a sequential loop to Atlantis reduced the median sweep time from 52.0 to 11.4 minutes. Weekly engineering time spent on drift fell from 9.4 to 2.4 hours, at an inference cost of about US\$0.08 per drifted project. The most common reasons for edited or rejected proposals were choosing the wrong kind of remediation and formatting an import identifier incorrectly.

Unique contribution to theory, practice and policy: The study gives evidence that a fixed workflow with one model call, a deterministic plan check, and mandatory human approval is a workable way to use language models in change-controlled infrastructure. For practitioners, it documents specific design choices and failure modes. For organizations writing policy on AI in operations, it supports keeping approval and apply rights with people while assigning analysis and drafting to the model.

Keywords: *Infrastructure as Code, Configuration Drift, Large Language Models, Atlantis, Site Reliability Engineering*

1. INTRODUCTION

Infrastructure as code (IaC) lets teams describe cloud resources in versioned text files and apply them with tools such as Terraform (Morris, 2025; Rahman et al., 2019). The approach works well as long as the files and the cloud agree. In practice they drift apart. An engineer changes a security group in the console during an incident, a security tool enables a bucket setting, a finance team adds cost tags, or a provider upgrade changes a default value. None of these edits pass through the repository, so the next plan shows differences that nobody working on the current change asked for (HashiCorp, n.d.-b).

Drift is rarely an outage by itself. The risk appears later. When an engineer plans an unrelated change, the output mixes the intended change with reversals of old manual edits. If the engineer applies it without reading carefully, Terraform may undo a fix that someone made on purpose. Practitioners list this kind of state mismatch among the recurring problems of maintaining IaC (Guerriero et al., 2019), and a recent study of state reconciliation defects shows how often IaC tools disagree with the systems they manage (Hassan et al., 2024).

At our organization, a US streaming television service, 142 Terraform root modules manage resources in three AWS accounts. Before this work, drift was found by a scheduled plan job and handled by whoever was on call. Engineers read the plan, decided whether each difference should be imported, written into the code, or reverted, and then wrote the change by hand. A timing study in the first quarter of 2026 put this work at about 9.4 engineering hours per week. This is the kind of repetitive, manual operational work that site reliability engineering practice treats as toil to be reduced (Beyer et al., 2016).

Large language models (LLMs) can now write and edit code well enough to take on routine parts of this task (Kon et al., 2024; Yang et al., 2024). Drift is a good candidate for several reasons. The input is structured, because Terraform can emit a plan as JSON. The output is structured as well, since it is HCL that Terraform can parse and plan. The unit of work is small, usually a single project. Most importantly, a proposed fix can be checked by planning it again, without changing any cloud resource.

This paper describes DriftAgent, the system we built around these properties, and reports twelve weeks of production use. Its contributions are as follows:

- A production architecture that combines parallel planning in Atlantis, one LLM call per drifted project, generated import blocks, merge requests for human review, and a daily Slack report.
- An argument, based on operating experience, for building such systems as fixed workflows rather than open-ended agent loops.
- Measurements of fix acceptance, planning time, review effort, and cost, together with a classification of failed proposals.

- Recommendations for engineering teams and for organizations that set policy on the use of AI in operations.

Section 2 reviews background and related work. Section 3 describes the system and the evaluation method. Section 4 reports the results, Section 5 discusses them, and Section 6 concludes with recommendations.

2. BACKGROUND AND RELATED WORK

2.1 Configuration Drift in Terraform

Terraform records the resources it manages in a state file. Before each plan it refreshes that state by reading the current attributes of every managed resource from the provider, and then compares the configuration, the refreshed state, and the recorded state (HashiCorp, n.d.-b). The JSON form of a plan keeps two relevant lists. The `resource_drift` list contains objects whose real attributes changed outside Terraform. The `resource_changes` list contains the actions Terraform would take to make the cloud match the configuration.

A team can resolve a difference in one of three ways. It can revert the cloud by applying the existing code, align the code with the cloud by editing HCL, or bring an unmanaged object under management by importing it. Since version 1.5, Terraform supports import blocks, which declare an import in configuration so that it is reviewed and planned like any other change (HashiCorp, 2023; HashiCorp, n.d.-a). This matters for automation, because an import can now be proposed in a merge request instead of being run as a command on an engineer's laptop.

2.2 Plan Automation with Atlantis

Atlantis is an open-source server that runs Terraform in response to merge request events and comments. It plans automatically when a merge request changes files in a project, posts the plan as a comment, and applies only when a reviewer comments atlantis apply and configured requirements such as approval are met (Atlantis, n.d.-b). A repository-level `atlantis.yaml` file defines projects as pairs of directory and workspace, and the `parallel_plan` option lets Atlantis plan many projects at once. Atlantis also offers an API endpoint that starts plans without a merge request (Atlantis, n.d.-a). This way of working keeps Git as the record of intended state, which is the central idea of GitOps (Beetz & Harrer, 2022).

2.3 Language Models in Infrastructure Operations

Research on LLM agents has moved from single prompts toward systems that reason and call tools in a loop (Schick et al., 2023; L. Wang et al., 2024; Yao et al., 2023). In software engineering, SWE-agent showed that a model given a suitable interface can edit repositories and resolve real issues (Yang et al., 2024). For IaC specifically, the IaC-Eval benchmark found that strong models often produce Terraform that fails to deploy or does not meet the stated intent, which is a reason to keep review and verification in the loop (Kon et al., 2024).

In cloud operations, most published LLM systems deal with incidents. Ahmed et al. (2023) evaluated LLM recommendations of root causes and mitigations for incidents at a large cloud provider. RCAgent applies a tool-augmented LLM agent to root cause analysis (Z. Wang et al., 2024), following earlier systems without language models such as CloudRCA (Zhang et al., 2021). AIOpsLab provides a framework for evaluating agents on operational tasks (Chen et al., 2025). Sculley et al. (2015) warned that machine learning components add maintenance costs of their own, which applies equally to a model placed inside an operations pipeline. Guidance from Anthropic separates workflows, in which code controls the sequence of steps, from agents, in which the model chooses its next action, and recommends the simpler form when it is sufficient (Schluntz & Zhang, 2024).

2.4 Research Gap

Detecting drift is well supported by existing tools. The published literature says much less about the step that takes most of an engineer's time: deciding how each difference should be resolved and writing the change. Incident-focused LLM systems act after something has failed, and IaC generation benchmarks start from a written requirement rather than from a plan diff. We found no production report on using an LLM to turn Terraform drift into reviewable code changes, including imports, inside an existing Atlantis and merge request workflow. This paper addresses that gap.

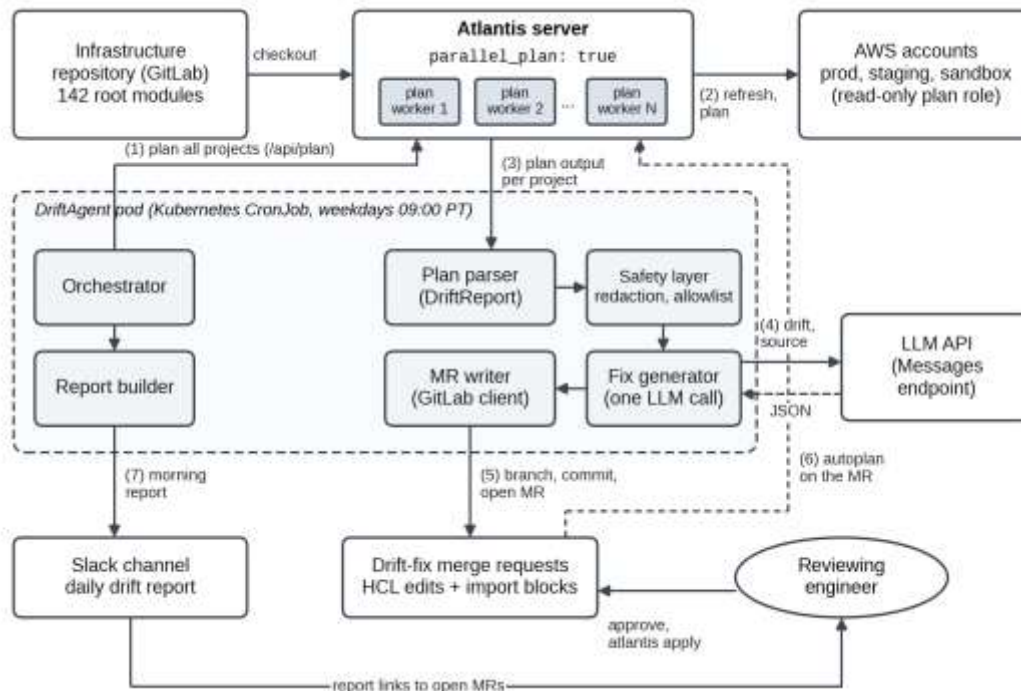
3. SYSTEM DESIGN AND METHODOLOGY

3.1 Design Goals

We set three goals. The first was to reduce the time an engineer spends on each drift event to a few minutes. The second was to make no infrastructure change without human approval. The third was to fit into tools the team already used, which meant GitLab merge requests, Atlantis, and Slack rather than a new dashboard. We also set explicit limits. The agent does not run apply, does not write to Terraform state, holds no cloud credentials beyond read access for planning, and keeps no plan output after a run ends.

3.2 Architecture Overview

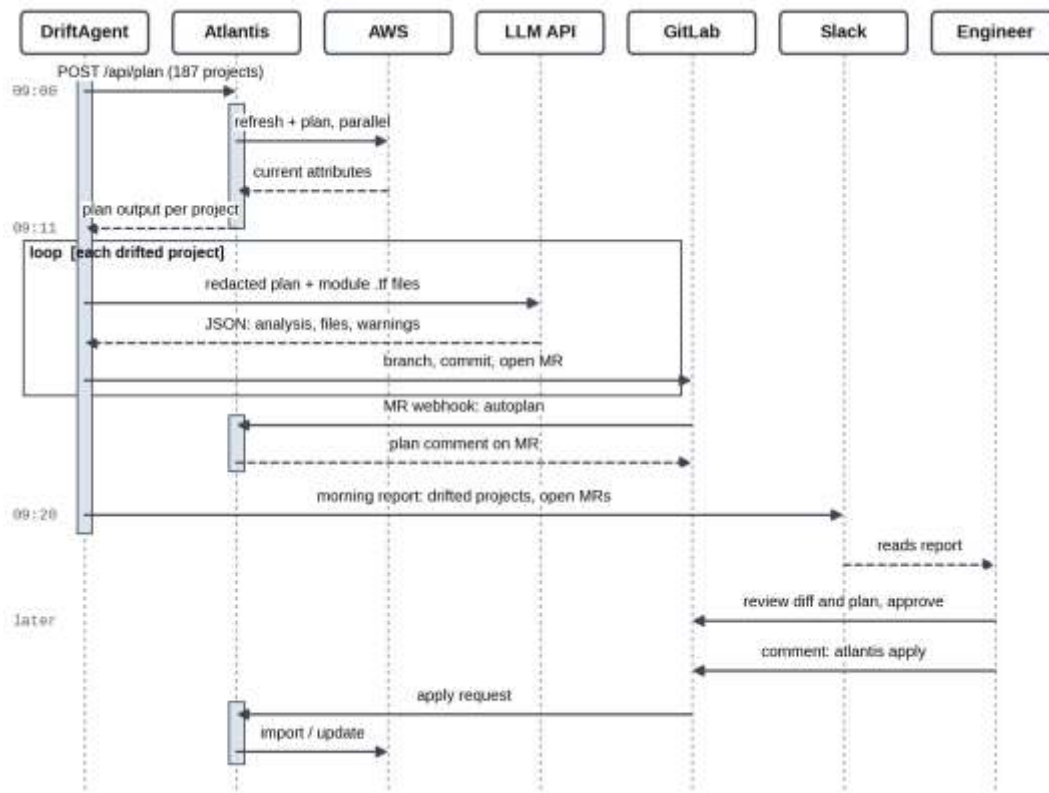
Figure 1 shows the main components and the order in which they interact during a run. DriftAgent is a Python 3.12 service packaged as a container and started by a Kubernetes CronJob at 09:00 Pacific time on weekdays. The job has a one-hour deadline and a Forbid concurrency policy, so only one run is active at a time. Inside the pod, an orchestrator drives a fixed sequence of steps. A plan parser turns each project's JSON plan into a DriftReport record. A safety layer redacts sensitive values and enforces command and file restrictions. A fix generator makes one call to the LLM for each drifted project. An MR writer creates the branch, commit, and merge request, and a report builder prepares the Slack summary. Planning itself runs on the existing Atlantis server, which reads the infrastructure repository and uses a read-only role in each AWS account.

**Figure 1: DriftAgent Architecture**

Numbers give the order of steps in a daily run. Solid arrows are requests made by a component, and dashed arrows are responses or events triggered by another system. The reviewing engineer is the only actor who can cause a change to be applied.

3.3 Daily Workflow

Figure 2 follows one run from start to finish. At 09:00 the agent sends a single plan request covering all projects. Atlantis refreshes and plans them in parallel against the AWS accounts and returns the results, usually by about 09:11. For each drifted project the agent calls the model, writes the returned files, and opens a merge request. Each new merge request triggers an Atlantis autoplan, and Atlantis posts the plan as a comment. When all projects have been handled, the agent posts the morning report to Slack, typically around 09:20. Engineers review the merge requests later in the day, and an approved change is applied through Atlantis.

**Figure 2: Sequence of a Daily Run**

The shaded bars show when each system is active. The loop runs once for every project whose plan contains drift. Steps below the report happen at a time chosen by the reviewer, often hours after the run.

3.4 Parallel Planning with Atlantis

The first version of the agent ran terraform init and terraform plan inside its own pod, one module after another. That design was simple, but it did not scale. With 142 modules, a full sweep took close to an hour and sometimes passed the job deadline (Table 3). It also duplicated settings that Atlantis already held, such as provider versions, backend configuration, and credentials for each workspace.

The current version hands planning to Atlantis. The repository's atlantis.yaml lists 187 projects, one for each combination of root module and workspace, and enables parallel planning (Listing 1). The agent sends one plan request for all projects through the Atlantis API (Listing 2). Atlantis runs the plans in a worker pool, which we limit to 15 concurrent plans, and returns output for every project. A custom workflow step also prints the JSON form of each plan, produced with terraform show -json, which the agent extracts and parses. Projects with no changes are recorded as clean. Projects whose plan fails are recorded as errors and reported, but they are not sent to the model.

Listing 1: Excerpt from atlantis.yaml (project names changed)

```

version: 3
parallel_plan: true
parallel_apply: false
projects:
  - name: networking-prod
    dir: terraform/networking
    workspace: prod
    apply_requirements: [approved, mergeable]
    autoplan:
      when_modified: ["*.tf", "../modules/**/*.*tf"]
  - name: networking-staging
    dir: terraform/networking
    workspace: staging
    apply_requirements: [approved, mergeable]

```

Listing 2: Requesting plans for all projects (simplified)

```

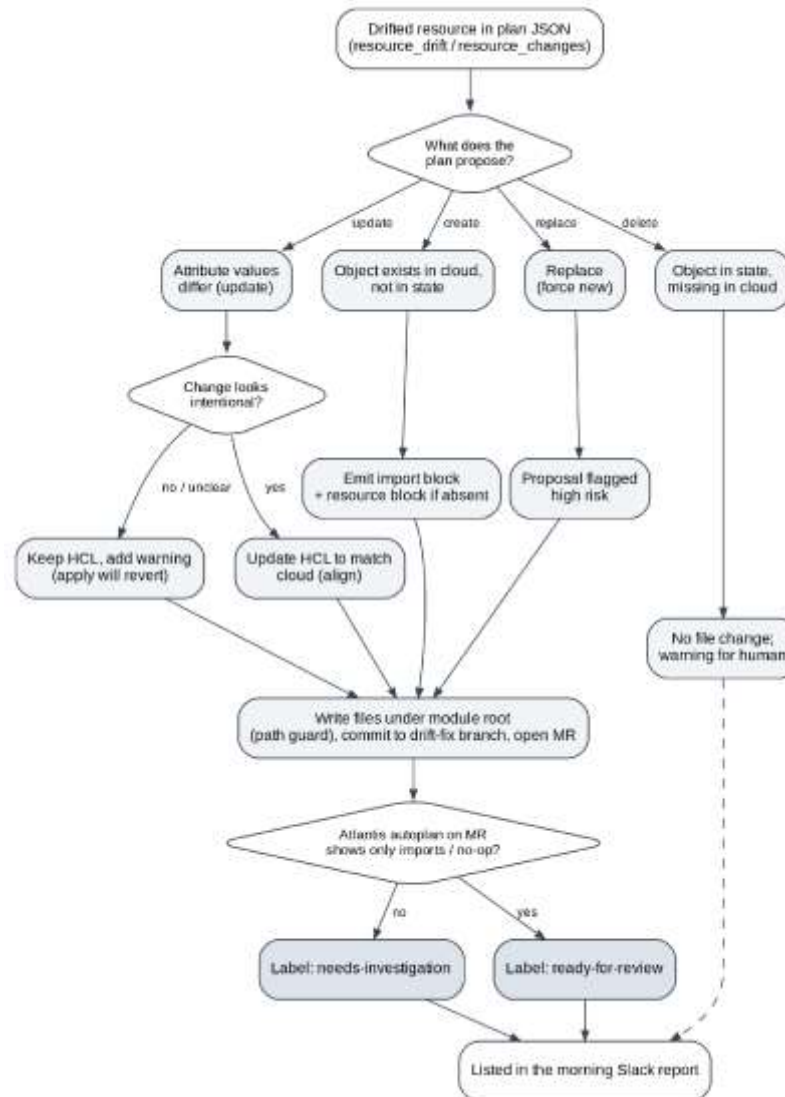
def plan_all(projects, cfg):
    body = {
        "Repository": cfg.repo,
        "Ref": "main",
        "Type": "Gitlab",
        "Paths": [{"Directory": p.dir, "Workspace": p.workspace}
                  for p in projects],
    }
    resp = requests.post(f"{cfg.atlantis_url}/api/plan", json=body,
                        headers={"X-Atlantis-Token": cfg.api_secret},
                        timeout=1800)
    resp.raise_for_status()
    return parse_project_results(resp.json())

```

3.5 Fix Generation and Import Blocks

For each drifted project the agent builds one prompt with four parts: the text plan, the lists of drifted and changed resources, the sanitized JSON plan, and the full contents of the project's .tf files. The text and JSON plans are each cut at 30,000 characters, with a marker that states how much was removed. The system prompt describes the model's role, the required response format, and a short list of rules. The model must produce only HCL and never shell commands, must not edit provider or backend blocks, and must follow the naming and style already used in the module. The response is a JSON object with three fields: a short analysis for reviewers, a list of files to write, and a list of warnings. Chain-of-thought prompting can help models on multi-step problems (Wei et al., 2022), but we ask only for a brief analysis, because reviewers read it in the merge request and a long reasoning trace made reviews slower in early trials.

Figure 3 summarizes how a drifted resource is handled. When an object exists in the cloud but not in state, the model writes an import block and, if the configuration lacks one, a matching resource block (Listing 3). When attribute values differ, the model judges whether the cloud value looks intentional. If it does, the model updates the HCL to match. If it does not, the model leaves the code alone and adds a warning that the next apply will revert the value. When an object is in state but missing from the cloud, the agent changes no files and asks a person to decide, since the deletion may have been deliberate. Planned replacements are still proposed but are flagged as high risk.

**Figure 3: Handling of a Drifted Resource**

The model makes the two judgments shown as diamonds in the upper half. The lower diamond is decided by the Atlantis plan on the merge request, not by the model.

Each file entry in the response names a path, an action (create, append, or replace_block), and the HCL content. Before writing, the fix writer resolves every path against the project directory and rejects any path that points outside it.

Listing 3: Import block generated for a security group created in the console (identifiers changed)

```
import {  
  to = aws_security_group.batch_workers  
  id = "sg-0a1b2c3d4e5f67890"  
}  
  
resource "aws_security_group" "batch_workers" {  
  name      = "batch-workers"  
  description = "Batch worker nodes"  
  vpc_id    = module.vpc.vpc_id  
  tags      = local.common_tags  
}
```

3.6 Merge Request Review

The agent opens one merge request per drifted project, on a branch named after the project and a timestamp. Separate merge requests let the owning team review its own change and prevent a problem in one project from delaying the others. The description contains the model's analysis, the affected resources, any warnings, the model identifier, a hash of the system prompt, and a review checklist.

Opening the merge request triggers an Atlantis autoplan, which is the main check on the model's work. If the fix is correct, the new plan contains only the proposed imports, or no changes at all. The agent reads this result and labels the merge request ready-for-review or needs-investigation. Reviewers see the diff and the Atlantis plan next to each other. Because every project requires an approved and mergeable merge request before Atlantis will apply (Listing 1), the model's output can reach the cloud only after a person approves it and comments atlantis apply.

3.7 Daily Slack Report

After the run, the agent posts one message to the team's Slack channel. Listing 4 shows an example with identifiers changed. The report gives the number of projects planned and how many were clean, drifted, or failed. It then lists each drifted project with its merge request and label, followed by the number of open drift merge requests and how many of them are older than three days. When nothing drifted and nothing failed, the report is a single line. Posting once a day at a predictable time replaced a stream of per-module alerts that engineers had learned to ignore.

Listing 4: Example morning report (identifiers changed)

Terraform drift report, Tue 16 Jun 2026 (sweep 09:00 to 09:12 PT)

Projects planned: 187 Clean: 179 Drifted: 7 Plan errors: 1

Drifted projects

networking-prod 3 resources MR !1412 ready-for-review

iam-roles-prod 5 resources MR !1413 ready-for-review

data-lake-staging 1 resource MR !1414 needs-investigation

(4 more)

Open drift MRs awaiting review: 5 (2 older than 3 days)

Plan errors: cdn-prod (provider timeout, will retry tomorrow)

3.8 Safety Controls

Several controls are enforced in code rather than by the prompt. Any Terraform command the agent runs locally passes through a helper that permits only init, plan, show, and version. Plan JSON is sanitized before it leaves the cluster: values stored under keys that match password, secret, token, key, credential, or private are replaced with a placeholder. File writes are limited to the project directory. The role Atlantis uses for drift plans is read-only, and apply requires approval. Credentials for GitLab, the LLM API, and Slack are kept in a Kubernetes Secret and passed to the pod as environment variables. We regard redaction as the weakest of these controls, because it depends on key names, and we review it whenever a new provider enters the repository.

3.9 Evaluation Method

We evaluated the system over twelve weeks, from May 4 to July 24, 2026. The agent ran on 60 weekdays. We excluded four runs that overlapped an Atlantis upgrade or a change freeze, which left 56. We defined a drift event as the first detection of a drifted project with a given set of changes. Later detections of the same unresolved drift were not counted again.

For each event we recorded the project, the resource changes, the model's latency and token counts, the Atlantis autoplan result, and the final state of the merge request. Outcomes came from merge request history: merged with no commits added by a reviewer, merged after reviewer commits, or closed. To check that these outcomes reflected the quality of the fixes, two engineers independently rated a random sample of 120 merge requests on a three-level scale. A proposal was rated acceptable if it resolved the drift correctly and could be merged as written, acceptable with edits if its remediation approach was correct but values, identifiers, or style needed changes, and

unacceptable if it chose the wrong remediation, would have reverted an intended change, or failed to plan. Inter-rater reliability was measured with Cohen's kappa (Cohen, 1960). The kappa value was 0.81, which indicates almost perfect agreement (Landis & Koch, 1977), and the ratings matched the history-based outcome in 92% of cases.

The baseline for engineering effort came from the first-quarter timing study, in which on-call engineers logged the time they spent on 71 drift events handled by hand. After deployment, review time was measured from GitLab activity timestamps and short self-reports. Planning time for the sequential design came from 25 runs of the first version between March 16 and April 24, 2026. Finally, we replayed stored plans for 40 projects, sampled to match the resource-type mix of the repository, to test two variations of the prompt and model.

The analysis used four primary measures: the outcome of each merge request, the share of merge requests labeled ready-for-review by the Atlantis autoplan, the time needed to plan all projects (sweep time), and engineering time per drift event. Token counts, model latency, and inference cost were recorded as secondary measures. Because sweep and review times were right-skewed, medians and 95th percentiles are reported alongside means, and outcomes are reported as counts and as percentages of all merge requests opened. No inferential statistical tests were applied, because the before and after periods differ in scope and the study describes a single deployment rather than a controlled experiment. Each edited or closed merge request was assigned one main failure reason, based on the reviewer's commits and comments, using the categories in Table 6. Finally, the findings were synthesized by comparing the merge request histories, the independent ratings, the timing data, and the replay results with one another.

4. RESULTS

4.1 Deployment Summary

Table 1 describes the scope of the deployment and the size of the evaluation data set.

Table 1: Deployment and Evaluation Summary

Item	Value
Terraform root modules	142
Atlantis projects (module and workspace)	187
AWS accounts	3
Evaluation window	12 weeks
Runs analyzed	56
Drift events	372
Merge requests opened	351
Events reported without a merge request	21

The 1 events without a merge request were objects missing from the cloud. For these the agent reports a warning instead of proposing a change.

4.2 Outcomes of Proposed Fixes

Table 2 shows what happened to the 351 merge requests the agent opened.

Table 2: Outcomes of Drift-Fix Merge Requests

Outcome	Merge requests	Percent
Merged without semantic changes	258	73.5%
Merged after reviewer edits	64	18.2%
Closed without merging	29	8.3%
Total	351	100%

The Atlantis autoplan labeled 309 merge requests (88.0%) ready-for-review and 42 (12.0%) needs-investigation. Of the 29 closed merge requests, 17 carried the needs-investigation label, so the plan check caught many weak proposals but not all of them. The median time from opening to merging was 1.3 business days.

4.3 Planning Time

Table 3 compares the time needed to plan every project under the first, sequential design and under Atlantis.

Table 3: Sweep Time, Sequential Planning Compared with Atlantis

Measure	Sequential	Atlantis
Runs measured	25	56
Projects planned per run	142	187
Median sweep time (minutes)	52.0	11.4
95th percentile sweep time (minutes)	63.1	16.8
Runs over the 60-minute deadline	4	0

Parallel planning cut the median sweep time by 78% while covering 45 more projects. The added projects are non-default workspaces, which the sequential version did not plan.

4.4 Review Effort and Cost

Table 4 compares engineering time per drift event before and after deployment, and Table 5 summarizes model usage.

Table 4: Engineering Time Spent on Drift

Measure	Before	After
Mean time per drift event (minutes)	18.2	4.6
Median time per drift event (minutes)	12	3
95th percentile (minutes)	64	14
Engineering time per week (hours)	9.4	2.4

Before deployment, time covered reading the plan, finding the module, writing the fix, and opening a merge request. After deployment, it covered reviewing the agent's merge request and, where needed, editing it. At about 31 drift events per week, the saving was roughly 7 engineering hours per week.

Table 5: Model Usage per Drifted Project

Measure	Mean	95th percentile
Input tokens	18,700	29,300
Output tokens	1,340	3,710
Model latency (seconds)	11.2	29.0
Cost (US\$)	0.08	0.14

Costs use list prices of US\$3 per million input tokens and US\$15 per million output tokens, without prompt caching. At the observed rate of drift, inference cost about US\$2.50 per week.

4.5 Failure Analysis

Table 6 classifies the 93 merge requests that reviewers edited or closed by the main reason the proposal fell short.

Table 6: Reasons Proposals Needed Edits or Were Closed

Reason	Count	Percent
Wrong remediation type (HCL edit instead of import, or the reverse)	27	29.0%
Intent misjudged (kept or reverted the wrong value)	20	21.5%
Import identifier in the wrong format	18	19.4%
Attribute not supported by the provider version	9	9.7%
Edited a provider or backend block	5	5.4%
Context lost to truncation	5	5.4%
Other	9	9.7%
Total	93	100%

Percentages are rounded. Import identifier errors fell from 13 in weeks 1–6 to 5 in weeks 7–12, after we added import identifier formats for common resource types to the prompt. All five edits to provider or backend blocks produced a failed autoplan and were closed.

In the replay of 40 stored plans, the production prompt produced acceptable fixes for 29 projects (72.5%). Removing the module's .tf files from the prompt lowered this to 19 (47.5%), mainly because the model no longer matched local variables, tags, and module references. A smaller and cheaper model produced acceptable fixes for 23 projects (57.5%) at about one quarter of the cost.

5. DISCUSSION

5.1 A Workflow Rather Than an Open Agent Loop

Our first prototype let the model call tools to plan, read files, and write files in a loop, in the style of ReAct (Yao et al., 2023). We replaced it with the fixed workflow described in this paper. The steps of a drift run are known in advance and depend only on plan results, so letting the model choose the next step added variation without adding ability. Restricting commands and file paths is simpler when Python code, not the model, issues them, and logs are easier to follow when every run has the same shape. This matches the advice to use the simplest structure that meets the need (Schluntz & Zhang, 2024). We still call the system an agent because it runs on its own schedule and produces changes without a person asking, but the model's part is limited to one step.

5.2 The Plan Check Does Most of the Verification

The Atlantis autoplan on each merge request does the job that a second, critic model does in some agent designs. It is deterministic, adds no inference cost, and uses the same code path that will later apply the change. Tasks without such a check, such as writing runbooks, may need a second model or closer human review. For IaC, a plan is almost always available, and a system should use it.

5.3 Import Blocks Need Resource-Specific Knowledge

Import blocks were the hardest part of the output to get right. Several of the failure reasons in Table 6 resemble the configuration data defects in the IaC defect taxonomy of Rahman et al. (2020), with the difference that here the faulty values were written by a model rather than a person. The model usually recognized when an import was needed, but the identifier format differs between resource types. Many resources use a plain identifier, while others, such as security group rules or IAM policy attachments, use composite strings whose format is defined by each provider resource. Adding a small table of identifier formats for the 30 resource types that appear most often in our repository removed most of these errors. We expect other teams will need a similar table for their own resource mix.

5.4 The Daily Report Changes Behavior

Engineers mentioned the Slack report more often than any other feature. Before, drift showed up as alerts at unpredictable times and was often put off. A single morning message with links to merge requests that were ready for review turned drift into a small, routine task at the start of the day. The count of merge requests older than three days also gave team leads a simple sign that review was falling behind.

5.5 Adoption and Trust

Building the first version took about four weeks of one engineer's time. Earning reviewers' trust took longer. Three practices helped: showing the model identifier and prompt hash in every merge request, reviewing closed merge requests together each week, and storing the system prompt in the repository so that prompt changes go through normal review. In hindsight, we would build redaction first rather than last, summarize long plans instead of cutting them, and version the prompt from the first day.

5.6 Limitations and Threats to Validity

The study covers one repository at one company, and its resource mix leans toward AWS networking and IAM. Results may differ for other cloud providers or for Kubernetes resources managed through Terraform. The baseline depends partly on self-reported times, which are imprecise. Reviewers could read the model's analysis, and a confident explanation may have made them more willing to approve. The independent ratings reduce this risk but do not remove it. Cost

figures use list prices, which change over time. Finally, the system cannot find cloud resources that no configuration describes, because a Terraform plan covers only declared resources. If the LLM API is unavailable, fix generation stops, but detection and the Slack report still work.

6. CONCLUSION AND RECOMMENDATIONS

6.1 Conclusion

We described DriftAgent, a system that plans all Terraform projects in parallel through Atlantis, uses one language model call per drifted project to draft HCL changes and import blocks, opens a merge request for each project, and reports the results in Slack every morning. Over twelve weeks, reviewers merged 91.7% of its merge requests, about four in five of those without changes, and the time engineers spent on drift fell by about three quarters at a very small inference cost. The results show that a language model can take over the drafting work in drift remediation while people keep control of what is applied.

6.2 Recommendations

Teams building similar systems should start with a fixed workflow and a single model call, run plans through an existing tool such as Atlantis rather than inside the agent, generate import blocks instead of running import commands, open one merge request per project, send one predictable daily report instead of many alerts, and include the module's source files and a table of import identifier formats in the prompt. For theory, the results suggest that when a task has structured input, structured output, and a cheap deterministic check, most of the value of a language model comes from drafting rather than from autonomy; future work could test this across other IaC tasks, such as policy violations, module refactoring, and provider upgrades, and across other organizations. For policy, organizations adopting AI in operations should keep approval and apply rights with people, record the model and prompt version behind every automated change, and give automated systems read-only credentials wherever possible.

ACKNOWLEDGMENTS

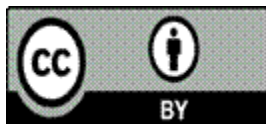
The author thanks the SRE and platform engineering teams at Philo who reviewed the first drift merge requests and whose feedback shaped the prompt and the daily report.

REFERENCES

- Ahmed, T., Ghosh, S., Bansal, C., Zimmermann, T., Zhang, X., & Rajmohan, S. (2023). Recommending root-cause and mitigation steps for cloud incidents using large language models. In *Proceedings of the 45th International Conference on Software Engineering* (pp. 1737-1749). IEEE. <https://doi.org/10.1109/ICSE48619.2023.00149>
- Atlantis. (n.d.-a). *API endpoints*. Retrieved September 24, 2026, from <https://www.runatlantis.io/docs/api-endpoints>

- Atlantis. (n.d.-b). *Repo level atlantis.yaml config*. Retrieved September 24, 2026, from <https://www.runatlantis.io/docs/repo-level-atlantis-yaml>
- Beetz, F., & Harrer, S. (2022). GitOps: The evolution of DevOps? *IEEE Software*, 39(4), 70-75. <https://doi.org/10.1109/MS.2021.3119106>
- Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (Eds.). (2016). *Site reliability engineering: How Google runs production systems*. O'Reilly Media.
- Chen, Y., Shetty, M., Somashekar, G., Ma, M., Simmhan, Y., Mace, J., Bansal, C., Wang, R., & Rajmohan, S. (2025). AIOpsLab: A holistic framework to evaluate AI agents for enabling autonomous clouds. *Proceedings of Machine Learning and Systems*, 7. https://proceedings.mlsys.org/paper_files/paper/2025/hash/d1f9e4a9f109b6e8b75ed362736f22ec-Abstract-Conference.html
- Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1), 37-46. <https://doi.org/10.1177/001316446002000104>
- Guerriero, M., Garriga, M., Tamburri, D. A., & Palomba, F. (2019). Adoption, support, and challenges of infrastructure-as-code: Insights from industry. In *2019 IEEE International Conference on Software Maintenance and Evolution* (pp. 580-589). IEEE. <https://doi.org/10.1109/ICSME.2019.00092>
- Hassan, M. M., Salvador, J., Santu, S. K. K., & Rahman, A. (2024). State reconciliation defects in infrastructure as code. *Proceedings of the ACM on Software Engineering*, 1(FSE), Article 83, 1865-1888. <https://doi.org/10.1145/3660790>
- HashiCorp. (n.d.-a). *Import block reference*. Terraform documentation. Retrieved September 24, 2026, from <https://developer.hashicorp.com/terraform/language/block/import>
- HashiCorp. (n.d.-b). *Manage resource drift*. Terraform tutorials. Retrieved September 24, 2026, from <https://developer.hashicorp.com/terraform/tutorials/state/resource-drift>
- HashiCorp. (2023, June 12). *Terraform 1.5 brings config-driven import and checks* [Blog post]. <https://www.hashicorp.com/en/blog/terraform-1-5-brings-config-driven-import-and-checks>
- Kon, P. T. J., Liu, J., Qiu, Y., Fan, W., He, T., Lin, L., Zhang, H., Park, O. M., Elengikal, G. S., Kang, Y., Chen, A., Chowdhury, M., Lee, M., & Wang, X. (2024). IaC-Eval: A code generation benchmark for cloud infrastructure-as-code programs. In *Advances in Neural Information Processing Systems 37: Datasets and Benchmarks Track*. https://proceedings.neurips.cc/paper_files/paper/2024/hash/f26b29298ae8acd94bd7e839688e329b-Abstract-Datasets_and_Benchmarks_Track.html
- Landis, J. R., & Koch, G. G. (1977). The measurement of observer agreement for categorical data. *Biometrics*, 33(1), 159-174. <https://doi.org/10.2307/2529310>
- Morris, K. (2025). *Infrastructure as code: Designing and delivering dynamic systems for the cloud age* (3rd ed.). O'Reilly Media.
- Rahman, A., Farhana, E., Parnin, C., & Williams, L. (2020). Gang of eight: A defect taxonomy for infrastructure as code scripts. In *Proceedings of the ACM/IEEE 42nd International*

- Conference on Software Engineering* (pp. 752-764). ACM. <https://doi.org/10.1145/3377811.3380409>
- Rahman, A., Mahdavi-Hezaveh, R., & Williams, L. (2019). A systematic mapping study of infrastructure as code research. *Information and Software Technology*, 108, 65-77. <https://doi.org/10.1016/j.infsof.2018.12.004>
- Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems 36* (pp. 68539-68551).
- Schluntz, E., & Zhang, B. (2024, December 19). *Building effective agents*. Anthropic. <https://www.anthropic.com/engineering/building-effective-agents>
- Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., & Dennison, D. (2015). Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems 28* (pp. 2503-2511).
- Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., & Wen, J. (2024). A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6), Article 186345. <https://doi.org/10.1007/s11704-024-40231-1>
- Wang, Z., Liu, Z., Zhang, Y., Zhong, A., Wang, J., Yin, F., Fan, L., Wu, L., & Wen, Q. (2024). RCAGENT: Cloud root cause analysis by autonomous agents with tool-augmented large language models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management* (pp. 4966-4974). ACM. <https://doi.org/10.1145/3627673.3680016>
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35* (pp. 24824-24837).
- Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., & Press, O. (2024). SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems 37*.
- Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the 11th International Conference on Learning Representations*. https://openreview.net/forum?id=WE_vluYUL-X
- Zhang, Y., Guan, Z., Qian, H., Xu, L., Liu, H., Wen, Q., Sun, L., Jiang, J., Fan, L., & Ke, M. (2021). CloudRCA: A root cause analysis framework for cloud computing platforms. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management* (pp. 4373-4382). ACM. <https://doi.org/10.1145/3459637.3481903>



©2026 by the Authors. This Article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>)