Poliac: Production Observability for LLM-based Infrastructure as Code

Haitam El Hayani
haitam.el-hayani@inria.fr
INRIA, IRISA, Université de Rennes, UMR 6074
Rennes, France

Jolan Philippe
jolan.philippe1@univ-orleans.fr
Université d'Orléans, INSA CVL, LIFO UR 4022
Orléans, France

Stéphanie Challita
stephanie.challita@inria.fr
INRIA, IRISA, Université de Rennes, UMR 6074
Rennes, France

ABSTRACT

Infrastructure as Code (IaC) enables the automated provisioning of cloud infrastructures, yet syntactically valid configurations may remain unsuitable for the applications they support. Addressing such issues requires knowledge of both the infrastructure and its runtime behavior. We introduce Poliac, an LLM-based framework that combines an IaC artifact, a natural-language specification, and aggregated operational data to generate infrastructure recommendations grounded in the state of the deployed system. We conduct a preliminary study on CorrectExam, a web application deployed on Grid'5000, collecting system- and application-level metrics under load. We query four LLMs with and without these operational aggregates. Without operational data, the models propose different resource modifications; when operational data are provided, all models recommend increasing vCPU while preserving memory, consistently with the observed CPU saturation and moderate memory utilization. These preliminary results suggest that operational data can reduce ambiguity in LLM-generated IaC recommendations and help avoid unnecessary resource provisioning.

ACM Reference Format:
Haitam El Hayani, Jolan Philippe, and Stéphanie Challita. 2026. Poliac: Production Observability for LLM-based Infrastructure as Code. In *Proceedings of CONFLANG'26*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Managing the underlying infrastructure that supports software applications has traditionally relied on manual processes. The Infrastructure as Code (IaC) paradigm addresses this limitation by representing infrastructure configurations as code, typically through declarative specifications [7, 13]. Provisioning tools, such as Terraform [8] and Pulumi [5], translate these configuration artifacts into API calls to cloud providers. However, writing such artifacts remains mainly manual and error-prone. Errors may first arise from syntactic defects, type inconsistencies, invalid references, or violations of predefined schemas and constraints. A configuration may also be syntactically valid and successfully provisioned but unsuitable for the application it supports. While the former can be addressed with static analysis [2, 9, 11], the latter requires knowledge of the application's behaviour combined with user-defined requirements. Existing frameworks for runtime system orchestration rely on contract-based decision logic driven by system metrics [3, 6]. However, the expressiveness of these contracts is constrained by the rule language supported by the orchestration engine. Consequently, requirements are quantitative and must remain general. Because distributed infrastructures are heterogeneous and involve multiple non-functional concerns, practitioners may also need to express qualitative, application-specific, or multi-objective requirements. Large Language Models (LLMs) offer an approach for addressing this limitation by interpreting requirements expressed in natural language and relating them to infrastructure code and configuration properties.

In this work, we introduce Poliac, a framework for linting infrastructure provisioning code against *natural-language specifications*. These specifications may capture operational constraints (e.g., bounds on resource usage), non-functional objectives (e.g., infrastructure cost), and higher-level requirements (e.g., migrating an infrastructure to another cloud provider with stronger commitments to energy sustainability). Such contracts may also combine several objectives, for example migrating an infrastructure while minimizing cost and favoring providers with stronger commitments to energy sustainability. To assist LLM-generated recommendations, Poliac complements the IaC artifact and natural-language specification with logs and operational data collected from the running infrastructure. This leads us to the following research question: *How does operational data support LLM-generated recommendations for Infrastructure as Code?*

The rest of the paper is organized as follows. Section 2 presents a motivating use case. Section 3 presents Poliac, its architecture and preliminary results. In Section 4, we present some related work. Finally, Section 5 concludes this work by opening to some perspectives.

2 MOTIVATING EXAMPLE

As a motivating example, consider a DevOps engineer who manages an AWS infrastructure using Terraform. The infrastructure hosts a web application that must support a large volume of user requests. Listing 1 shows an excerpt of the Terraform manifest containing three resources: an `aws_ami` data source that identifies the machine image used by the application, an `aws_subnet` resource that defines the private network in which the application is deployed, and an `aws_instance` resource that provisions the EC2 instance hosting the web application. Under high query loads, the capacity of

CONFLANG'26, October 12–16, 2026, Munich, Germany

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Listing 1: Excerpt of a Terraform configuration for AWS

```
data "aws_ami" "base" { ... }
resource "aws_subnet" "private" { ... }
resource "aws_instance" "web_app" {
  ami           = data.aws_ami.base.id
  subnet_id     = aws_subnet.private.id
  instance_type = "m9g.xlarge"
}
```

> Here is a Terraform configuration for a deployed service. The web application is missing its latency Service Level Objective (SLO) of p95 latency < 200 ms under peak load. Update the Terraform so that the SLO is met. Give the direct code changes and explain your reasoning.

Prompt 1: Functional specification given to the LLM in the motivating example.

web_app becomes the main performance bottleneck. We therefore focus on this resource in the remainder of the motivating example.

The configuration successfully passes `terraform validate`, produces a clean `terraform plan`, and triggers no findings from `tflint`. However, after one week, the application experiences a growth of its user base, and the 95th percentile of the measured latency ($p95$) exceeds the service level objective (SLO) of 200ms.

Because the engineer is neither a systems expert nor familiar with the application, identifying the appropriate infrastructure changes is challenging. The engineer therefore uses LLMs for assistance and provides the Terraform configuration together with Prompt 1.

However, the LLMs propose different modifications for the same configuration and objective. GPT-5.5 recommends scaling the instance vertically from `m9g.xlarge` to `m9g.2xlarge`, and DeepSeek-V4-Flash proposes the larger `m9g.4xlarge`. Gemini-3.6-Flash instead keeps the instance size unchanged and introduces an additional resource, `aws_autoscaling_group`, to automatically adjust the number of EC2 instances according to application demand, whereas Sonnet changes the instance family to `c7g.xlarge` and also introduces autoscaling. Although each proposal is plausible, the prompt alone provides no empirical data for selecting among competing suggestions. As a consequence, the proposed solution may over-provision the infrastructure, leading to unnecessary costs (operational and financial). Under AWS on-demand pricing, the originally deployed `m9g.xlarge` costs approximately $141 per month while `m9g.2xlarge` and `m9g.4xlarge` respectively cost $282 and $564 per month. The proposed `c7g.xlarge` costs less, with $104 per instance, but the overall price may exceed other proposals depending on the number of instances deployed within the auto-scaling group.

Model	Suggested fix
GPT-5.5	Increase instance size to `m9g.2xlarge`
DeepSeek-V4-Flash	Increase instance size to `m9g.4xlarge`
Gemini-3.6-flash	Keep instance size, and add autoscaling group
Sonnet-5	Change instance family to `c7g.xlarge` and add autoscaling group

Table 1: LLMs responses to artifact-only prompt

3 POLIAC

In this paper, we introduce Poliac, an LLM-based framework to assist DevOps engineers writing IaC scripts using operational data. Its core idea is to leverage two complementary assets for improving an IaC artifact: (i) LLMs for specifying a natural-language specification, using their ability to reason over code and natural-language requirements, and (ii) operational data, which capture the state and behavior of the deployed system. As a result, Poliac uses linters to suggest improvements to DevOps engineers writing IaC artifacts.

As illustrated in Figure 1, Poliac provides an LLM with three inputs: (i) the IaC artifact describing the deployed infrastructure, (ii) an aggregated view of the operational data produced by the deployed system, and (iii) a specification in which the engineer specifies both functional and non-functional requirements and optimization objectives in natural language.

Combining these three sources of information provides additional runtime context to LLMs about the deployed system, enabling them to better identify the infrastructure change that addresses the observed issue. As illustrated by the motivating example in Section 2, the IaC artifact and the prompt alone provide insufficient information and may lead to several potential, sometimes conflicting, modification proposals. By incorporating operational data, Poliac aims to reduce this ambiguity and guide the LLM toward a converged recommendation that is consistent with the observed behavior of the deployed system.

The operational data and the specification play complementary roles. On one side, the deployed infrastructure and the application running on it continuously produce operational data: system metrics (*e.g.,* CPU and memory usage), and application-level metrics (*e.g.,* latency and throughput). These measurements provide evidence about the load and performance of the deployed system. Rather than supplying raw data to the LLMs, Poliac first computes statistical aggregates such as percentiles, usage distributions, and saturation indicators. These aggregates can be obtained from the cloud provider's monitoring service or from a monitoring stack such as Prometheus[1] or Grafana[2]. Determining which aggregates are necessary and sufficient is left for future work. On the other side, the specification captures the DevOps' intent using natural language. It can specify functional requirements, relying on direct operational measurements, but also on non-functional objectives (e.g., limits on financial costs). Investigating more structured specification representations is left for future work.

Preliminary results. We experimented with Poliac on aggregated operational data from deployments of CorrectExam[3], a microservice web application for exam grading, running under several

[1] https://prometheus.io/
[2] https://grafana.com/
[3] https://correctexam.github.io/

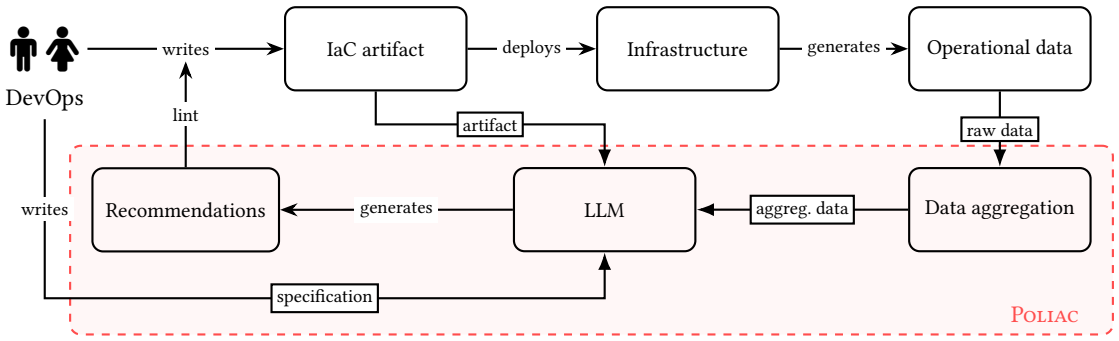

Figure 1: Overview of Poliac. The IaC artifact deploys the infrastructure, whose operational data is aggregated and supplied to the model together with the artifact and the engineer's specification.

infrastructure configurations on Grid'5000[4], a large-scale experimental platform. We report the configuration provisioned with 2 vCPUs and 4 GB of memory, using the `libvirt` provider. Under this configuration, the application sustains up to 200 concurrent users within its latency objective of 200ms; beyond that threshold, latency degrades progressively. Prototype implementation, operational data, and LLMs responses are publicly available on GitHub[5].

We collected and aggregated several metrics: CPU, memory, disk, network, latency, and throughput. We submitted Prompt 2 to four LLMs (*i.e.,* GPT-5.5, DeepSeek-V4-Flash, Gemini-3.6-flash, Sonnet-5), each with and without the aggregated data, holding all other inputs constant. Table 2 reports the outcome for each LLM. Without operational data the models disagree: GPT-5.5 proposes increasing vCPU alone, while the remaining three increase both vCPU and memory, to 4 vCPUs and 8 GB. Once the aggregates are supplied, all four converge on increasing vCPU alone. The LLMs tend to naively improve the overall computation capacity by both increasing the CPU and memory. However, the aggregates show that memory utilisation remains between 39% and 56% across the observation window, never approaching saturation, while CPU reaches approximately 99% at peak load. Hence, under the same workload, additional vCPUs relieve the observed constraint, whereas additional memory provisions a resource that was never under pressure.

Additional fine-grained metrics, such as CPU energy consumption over time, could provide a more detailed characterization of processor behavior and help the LLM distinguish between different causes of resource saturation, leading to better recommendations for modifying the infrastructure.

> The application is experiencing increased latency. Solve the issue without over-provisioning the deployed resources:
>
> - Primary goal: Suggest the IaC change to reduce the application's latency.
> - The latency increases under peak load (starting from 500 users), solve it while staying resource-efficient, don't over-provision the infrastructure in order not to have any wasted resources.

Prompt 2: Functional specification for improving CorrectExam performance using Poliac.

Model	Suggestion without aggregated data
GPT-5.5	Increase vCPU to 4
DeepSeek-V4-Flash	Increase vCPU to 4 and memory to 8GB
Gemini-3.6-flash	Increase vCPU to 4 and memory to 8GB
Sonnet-5	Increase vCPU to 4 and memory to 8GB
Model	**Suggestion with aggregated data**
GPT-5.5	Increase vCPU to 4
DeepSeek-V4-Flash	Increase vCPU to 4
Gemini-3.6-flash	Increase vCPU to 4
Sonnet-5	Increase vCPU to 4

Table 2: Comparison between LLMs' suggestions based on the supplied aggregated operational data

4 RELATED WORK

Existing work primarily focuses on properties that can be derived statically from IaC artifacts, mostly focusing on quality issues for configuration management [4, 14]. Similarly, additional works aimed at enhancing code quality for provisioning IaC. TerraMetrics extracts code-level metrics from Terraform manifests to assess code quality and support maintainability analysis [2], while Häyhä analyses changes between infrastructure deployments to detect intra-update sniping vulnerabilities [11]. In contrast, dynamic-data-driven approaches such as CherryPick [1] and GPTuner [10] tune configuration parameters by combining workload observations with Bayesian analysis. However, these approaches do not support contracts expressed in natural language, thereby limiting the expression of non-functional properties. Pulumi Neo[6] addresses this limitation by providing an AI agent for managing Pulumi-based infrastructures. It supports natural-language interactions for generating code, diagnosing deployment problems, checking policies, optimizing costs, and executing infrastructure changes. A similar approach has been applied to configuration management, another capability of IaC tools, through Ansible Lightspeed [16]. Unlike

[4]https://www.grid5000.fr/
[5]https://github.com/DEVhaitam/POLIAC
[6]https://www.pulumi.com/product/neo/

existing approaches, Poliac aims at both using operational data to assess the compliance of IaC provisioning artifacts and using quantitative and qualitative requirements expressed using natural language.

5 DISCUSSION

Limitations and threats to validity. Our current work has several limitations. First, our prototype targets `libvirt` resources provisioned with Terraform manifests, and the preliminary results only considered a narrow set of attributes: vCPU and memory. We voluntarily excluded additional infrastructure changes, such as introducing autoscaling or load balancing, since no validation mechanism of recommendation is implemented yet. Second, our preliminary tests rely on operational data coming from a test environment and are conducted on an experimental platform, which may not fully capture the behavior of production systems. To approximate an AWS deployment, we assume a correspondence between Grid'5000 resources and AWS instance types, assessing the potential impact of Poliac recommendations. This assumption may limit the external validity of our results, and further experiments on real production infrastructures are needed for confirmation. Third, Poliac currently consumes pre-aggregated operational metrics from CSV files. While aggregation reduces the amount of data given to the LLM, it may hide short-lived events or temporal patterns that are relevant to infrastructure decisions. Conversely, in settings such as nightly CI/CD pipelines, immediate recommendations are not required. Longer processing times may therefore be acceptable, potentially allowing Poliac to analyze larger volumes of raw operational data without prior aggregation. Finally, Poliac inherits the limitations of LLM-generated code, including non-deterministic outputs, API latency, outdated knowledge, hallucinated resources or attributes, invalid attribute combinations, and configurations that may fail to deploy [12]. The current prototype should consider additional knowledge to provide to LLMs, such as resource catalogs (*i.e.,* specification, resource schemas) and compatibility constraints.

Future work. A first direction concerns the operational context supplied to the model. Beyond system and application metrics such as CPU utilization, latency, or throughput, future versions of Poliac could incorporate IaC-specific metrics, including properties on deployment such as duration, stability, or security [17]. A second direction concerns the scope and granularity of the user-defined specification. LLMs are particularly attractive when requirements span several abstraction levels or combine objectives that are difficult to encode as fixed orchestration rules, or jointly considering performance, security, and sustainability objectives. Future work should therefore investigate how such specifications should be structured and how multiple objectives should be prioritized. In addition, we plan to study the impact of LLM inference parameters, particularly temperature [15], on the consistency of the generated infrastructure recommendations. Finally, recommendation reliability could be improved by comparing different LLMs, grounding them in up-to-date provider information and introducing explicit validation with ranking mechanisms before recommendations are presented to engineers.

This work is funded by the National Research Agency under the France 2030 program (ANR-23-PECL-0008).

REFERENCES

[1] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. 2017. Cherrypick: adaptively unearthing the best cloud configurations for big data analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (Boston, MA, USA) *(NSDI'17)*. USENIX Association, USA, 469–482.

[2] Mahi Begoug, Moataz Chouchen, and Ali Ouni. 2024. TerraMetrics: An Open Source Tool for Infrastructure-as-Code (IaC) Quality Metrics in Terraform. In *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension* (Lisbon, Portugal) *(ICPC '24)*. Association for Computing Machinery, New York, NY, USA, 450–454. https://doi.org/10.1145/3643916.3644439

[3] Daniel Casini, Paolo Pazzaglia, and Matthias Becker. 2025. Managing real-time constraints through monitoring and analysis-driven edge orchestration. *J. Syst. Archit.* 163, C (June 2025), 16 pages. https://doi.org/10.1016/j.sysarc.2025.103403

[4] Michele Chiari, Michele De Pascalis, and Matteo Pradella. 2022. Static Analysis of Infrastructure as Code: a Survey. In *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*. 218–225. https://doi.org/10.1109/ICSA-C54293.2022.00049

[5] Pulumi Corporation. 2017. Pulumi. https://www.pulumi.com/.

[6] Osama Mohammed Dighriri, Priyadarsi Nanda, Manoranjan Mohanty, Bashair Alrashed, and Ibrahim Haddadi. 2025. SecuRecNet-IoT: Adaptive Secure Reconfiguration and Session-Aware Communication in IoT Edge Networks. In *2025 IEEE 24th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE Computer Society, Los Alamitos, CA, USA, 3097–3102. https://doi.org/10.1109/Trustcom66490.2025.00369

[7] Michele Guerriero, Martin Garriga, Damian A. Tamburri, and Fabio Palomba. 2019. Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, Cleveland, OH USA, 580–589. https://doi.org/10.1109/ICSME.2019.00092

[8] HashiCorp. 2014. Terraform. https://developer.hashicorp.com/terraform.

[9] Hanyang Hu, Yani Bu, Kristen Wong, Gaurav Sood, Karen Smiley, and Akond Rahman. 2023. Characterizing Static Analysis Alerts for Terraform Manifests: An Experience Report. In *2023 IEEE Secure Development Conference (SecDev)*. IEEE, Atlanta, GA USA, 7–13. https://doi.org/10.1109/SecDev56634.2023.00014

[10] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Mingjie Tang, and Jianguo Wang. 2024. GPTuner: A Manual-Reading Database Tuning System via GPT-Guided Bayesian Optimization. *Proceedings of the VLDB Endowment* 17, 8 (2024), 1939–1952. https://doi.org/10.14778/3659437.3659449

[11] Julien Lepiller, Ruzica Piskac, Martin Schäf, and Mark Santolucito. 2021. Analyzing Infrastructure as Code to Prevent Intra-update Sniping Vulnerabilities. In *Tools and Algorithms for the Construction and Analysis of Systems*, Jan Friso Groote and Kim Guldstrand Larsen (Eds.). Springer International Publishing, Cham, 105–123.

[12] Roman Nekrasov, Stefano Fossati, Indika Kumara, Damian Andrew Tamburri, and Willem-Jan van den Heuvel. 2026. IaC Generation with LLMs: An Error Taxonomy and A Study on Configuration Knowledge Injection. https://doi.org/10.1145/3817608 Just Accepted.

[13] Akond Rahman, Rezvan Mahdavi-Hezaveh, and Laurie Williams. 2019. A Systematic Mapping Study of Infrastructure as Code Research. *Information and Software Technology* 108 (2019), 65–77.

[14] Pandu Ranga Reddy Konala, Vimal Kumar, and David Bainbridge. 2023. SoK: Static Configuration Analysis in Infrastructure as Code Scripts. In *2023 IEEE International Conference on Cyber Security and Resilience (CSR)*. 281–288. https://doi.org/10.1109/CSR57506.2023.10224925

[15] Matthew Renze. 2024. The effect of sampling temperature on problem solving in large language models. In *Findings of the association for computational linguistics: EMNLP 2024*. 7346–7356.

[16] Priyam Sahoo, Saurabh Pujar, Ganesh Nalawade, Richard Genhardt, Louis Mandel, and Luca Buratti. 2024. Ansible Lightspeed: A Code Generation Service for IT Automation. In *39th IEEE/ACM International Conference on Automated Software Engineering* (Sacramento, CA, USA) *(ASE '24)*. Association for Computing Machinery, New York, NY, USA, 2148–2158. https://doi.org/10.1145/3691620.3695277

[17] Mohamed Haady Tiemtore and Frédéric Loulergue. 2026. Tools for Detecting Security Issues in Infrastructure-as-Code: A Focused Literature Review. In *Empirical Software Engineering and Measurement (ESEM)* (München, Germany) *(LIPIcs)*. Dagstuhl Publishing. https://doi.org/10.4230/LIPIcs.ESEM.2026.54 Emerging Results, Vision, and Reflection Papers Track, to appear.