Infrastructure Intelligence Agent:

An AI-Powered Operations Co-Pilot for Cloud Infrastructure Management

Marcin Wypyszynski

Cloud & Navigation Platform Engineering

marcin.wypyszynski@gmail.com

Version 1.0 – April 2026

Abstract: This paper presents the design, implementation, and production deployment of an *Infrastructure Intelligence Agent*—an AI-powered system that enables natural-language interaction with cloud infrastructure. The agent leverages a large language model (Amazon Nova Pro via AWS Bedrock) combined with a tool-use agentic loop to autonomously query AWS APIs, analyze Terraform configurations, correlate metrics, and deliver actionable operational insights. The system implements 53 runtime tools across 10 operational domains including FinOps optimization, reliability monitoring, drift detection, incident investigation, and Terraform productivity. A novel adaptive learning subsystem enables the agent to improve over time through auto-extracted insights, service baseline tracking, user feedback incorporation, and reusable playbook generation. Deployed in production managing infrastructure across 4 environments, 2 AWS regions, and 2 accounts, the agent operates at $7–23/month while providing capabilities that replace hours of manual cross-service investigation. Security is enforced through read-only defaults, approval-gated write operations, and a complete audit trail. We detail the architecture, tool dispatch mechanism, learning system, security model, and production evaluation results.

Keywords: AI agent, infrastructure management, cloud operations, tool-use, LLM, FinOps, autoscaling, drift detection, adaptive learning, AWS

1 Introduction

Managing modern cloud infrastructure involves navigating a fragmented landscape of services, consoles, and observability tools. A typical investigation—determining why a service is degraded—requires context-switching between Amazon ECS, CloudWatch Metrics, CloudWatch Logs Insights, Cost Explorer, and Terraform configuration files. This cognitive overhead grows linearly with environment count and service count.

We present the *Infrastructure Intelligence Agent*, a system that unifies these fragmented data sources behind a natural language interface. Instead of writing queries, navigating dashboards, or correlating data manually, an engineer asks:

> *"Which services in pre-production are over-provisioned and how much can we save?"*

The agent autonomously determines which tools to invoke, executes them against live AWS APIs, reasons over intermediate results, and delivers a structured, actionable response.

1.1 Contributions

This paper makes the following contributions:

1. **Agentic architecture for infrastructure management:** A production-grade system using the Bedrock Converse API tool-use protocol with 53 registered tools spanning 10 operational domains.
2. **Adaptive learning system:** A novel subsystem comprising four components—Learning Engine, Baseline Tracker, Feedback Collector, and Playbook Store—that enables continuous improvement without model fine-tuning.

3. **Security-first design:** A governance model with read-only defaults, explicit approval gates for write operations, secret masking, path traversal protection, and complete audit trail.

4. **Production evaluation:** Deployment results across 4 environments, 2 regions, and 9+ ECS Fargate services demonstrating zero-error operation at $7–23/month.

2 Related Work

2.1 AI for Cloud Operations (AIOps)

The AIOps paradigm [1] advocates using machine learning for IT operations, typically focusing on anomaly detection and root cause analysis. Systems like Microsoft's DeepLog [2] and CloudRCA [3] apply deep learning to log analysis and root cause identification. Our approach differs fundamentally: rather than building specialized ML models, we leverage a general-purpose LLM as a reasoning engine that dynamically composes tool calls to investigate infrastructure state.

2.2 LLM-Based Agents

The emergence of tool-using LLM agents [4, 5] has demonstrated that language models can effectively invoke external APIs. Systems like AutoGPT [6] and LangChain [7] provide general-purpose agent frameworks. Our work applies this paradigm specifically to infrastructure operations with domain-specific tools, safety constraints, and an adaptive learning system designed for operational continuity.

2.3 Infrastructure as Code Intelligence

Tools like Terraform [8], Pulumi, and AWS CDK manage infrastructure declaratively. Our agent complements these by detecting drift between desired and actual state, generating configuration changes, and providing a natural-language interface to infrastructure-as-code workflows.

3 System Architecture

3.1 Overview

The system consists of five architectural layers (Figure 1):

1. **Interface Layer:** CLI (Click + Rich), AWS Lambda, or EventBridge scheduled invocations.

2. **Orchestrator:** Central coordinator managing the conversation loop with the LLM via the Bedrock Converse API.

3. **Tool Layer:** 53 registered tools organized into 10 domains, each implementing a standardized interface.

4. **Learning Layer:** Adaptive components for knowledge extraction, baseline tracking, feedback processing, and playbook management.

5. **Storage Layer:** DynamoDB tables for knowledge, conversations, baselines, and feedback; S3 for report archival.

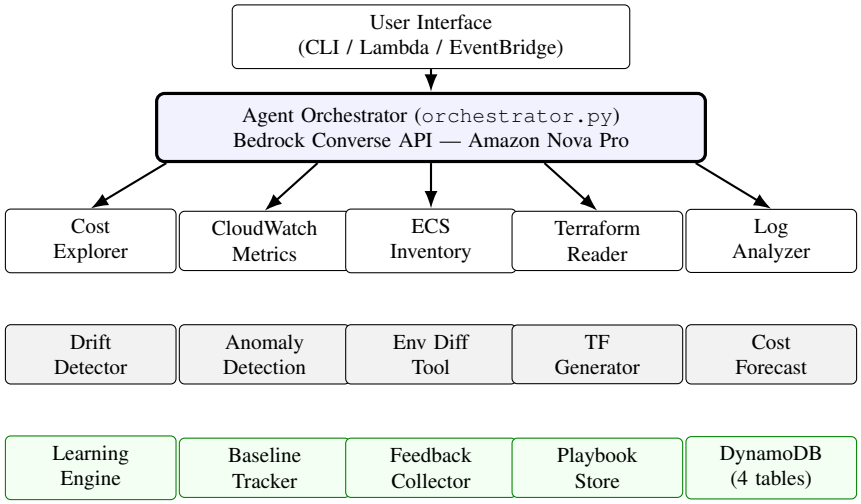

Figure 1: System architecture showing the five layers: interface, orchestrator, primary tools, capability tools, and the adaptive learning system.

3.2 Agent Orchestrator

The orchestrator implements the core reasoning loop:

1. **Prompt Construction:** Assembles the system prompt from a base template, retrieved memories, corrections, playbooks, and detected anomalies.
2. **Tool-Use Loop:** Sends the user query plus tool definitions to the LLM via the Bedrock Converse API. The model selects tools to invoke; the orchestrator executes them and returns results. This loop continues for up to 30 iterations.
3. **Learning Hook:** After each tool execution, the Learning Engine asynchronously extracts insights from the result.
4. **Playbook Generation:** If $\geq$3 tools were used in a successful investigation, the workflow is automatically saved as a reusable playbook.
5. **Response Delivery:** The final structured Markdown response is rendered via the Rich terminal UI, sent to MS Teams, or archived to S3.

The bounded iteration limit ($N_{\max} = 30$) ensures cost and latency control while allowing sufficiently deep investigations.

3.3 Tool Dispatch Mechanism

Each tool is registered with the Bedrock Converse API using a JSON Schema definition specifying its name, description, and input parameters. The LLM selects tools based on the user's intent and intermediate evidence. Tool execution follows a standardized pattern:

```
def execute_tool(tool_name, tool_input):
    """Dispatch tool call with error handling and audit."""
    validate_input(tool_name, tool_input)
    result = TOOL_REGISTRY[tool_name](tool_input)
    audit_log(tool_name, tool_input, result)
    learning_engine.auto_learn(tool_name, result)
    return result
```

4 Capability Domains

The 53 tools are organized into 10 operational domains (Table 1).

Table 1: Tool domains and representative capabilities

Domain	Tools	Representative Capability
FinOps Optimization	5	Cost breakdown, rightsizing savings, forecast, simulation
Reliability Monitoring	4	Health checks, anomaly detection, proactive alerting
Drift & Compliance	4	Terraform vs actual state comparison, severity classification
Log Intelligence	6	Error clustering, OOM/timeout detection, spike analysis
Incident Investigation	3	Timeline reconstruction across deployments, metrics, logs
Dependency & Blast Radius	6	Service dependency graph, change impact estimation
Network & Queue Visibility	13	VPC, security groups, load balancers, SQS/DLQ health
Terraform Productivity	4	Config generation, plan/apply workflow with approval gates
Knowledge & Learning	8	Memory store, corrections, baselines, playbook retrieval
Executive Reporting	–	Aggregated reports with actions, not raw telemetry

4.1 FinOps Optimization

The agent correlates three data sources to identify optimization opportunities:

1. **Cost Explorer API:** Per-service cost attribution via resource tags.
2. **ECS Inventory:** Current task count, CPU, and memory allocation.
3. **CloudWatch Metrics:** 14-day CPU and memory utilization (average, P99, min).

Services where average utilization is below a configurable threshold (default: 40%) are flagged as over-provisioned. The agent calculates dollar savings using AWS Fargate pricing models and can generate Terraform changes to implement recommendations.

4.2 Drift Detection

The drift detector compares Terraform desired state with live AWS state:

$$\text{drift}(s) = \text{State}_{\text{terraform}}(s) \oplus \text{State}_{\text{aws}}(s) \tag{1}$$

where $\oplus$ denotes the symmetric difference operator applied to configuration parameters (image tag, CPU, memory, environment variables, task count). Drift items are classified by severity:

- **CRITICAL:** Image tag mismatch (hotfix deployed but not in Terraform).
- **HIGH:** Resource allocation mismatch (CPU/memory).
- **MEDIUM:** Environment variable or configuration differences.
- **LOW:** Cosmetic differences (labels, tags).

4.3 Incident Investigation

For incident triage, the agent reconstructs a correlated timeline by invoking multiple tools:

1. `get_deployment_history` — recent deployments and rollout state.
2. `get_service_utilization` — CPU/memory metrics around the incident window.
3. `search_logs` — error patterns in CloudWatch Logs Insights.
4. `detect_log_anomalies` — automated spike detection (OOM, timeouts, circuit breakers).
5. `get_queue_health` — SQS/DLQ depth and age metrics.

The LLM correlates these results to identify root cause and recommend remediation, reducing mean-time-to-diagnosis from hours of manual investigation to minutes of agent-guided analysis.

5 Adaptive Learning System

A distinguishing feature of the agent is its ability to improve over time without model fine-tuning. The learning system comprises four components stored in DynamoDB.

5.1 Learning Engine

After each tool execution, the Learning Engine automatically extracts operational insights:

- Service health patterns (e.g., "Service X consistently runs at $<$15% CPU").
- Cost anomalies (e.g., "Cost increased 40% week-over-week").
- Error signatures (e.g., "TLS timeout pattern on DocumentDB connections").

These insights are stored as typed memories in DynamoDB with deduplication based on semantic similarity. They are injected into subsequent system prompts, enabling the agent to reference prior findings.

5.2 Baseline Tracker

The Baseline Tracker periodically snapshots key metrics for each service:

$$B_s(t) = \left\{\overline{\text{CPU}}_s(t),\ \overline{\text{Mem}}_s(t),\ C_s(t),\ N_s(t)\right\} \tag{2}$$

where $\overline{\text{CPU}}_s(t)$ and $\overline{\text{Mem}}_s(t)$ are 14-day rolling averages, $C_s(t)$ is the monthly cost, and $N_s(t)$ is the task count for service s at time t.

Deviations exceeding configurable thresholds from the rolling baseline trigger anomaly flags that are surfaced in the agent's system prompt and in proactive monitoring reports.

5.3 Feedback Collector

When the agent produces recommendations (e.g., rightsizing proposals), users can:

- **Approve:** Confirms the recommendation was correct, increasing confidence for similar future suggestions.
- **Reject:** Marks the recommendation as incorrect, with an optional reason.
- **Correct:** Provides the right answer, which is stored as a correction memory and injected into future prompts to prevent the same mistake.

5.4 Playbook Store

When a successful investigation uses $\geq$3 tools, the sequence of tool calls and their parameters is automatically saved as a playbook. On subsequent similar queries, the agent retrieves relevant playbooks and can follow proven investigation paths, resulting in faster and more consistent responses.

6 Security Model

Security is designed into the architecture, not added as an afterthought.

Table 2: Security controls

Control	Implementation
Read-only default	`AGENT_READ_ONLY=true`; all write operations blocked unless explicitly enabled
Approval gates	Terraform apply and file modification require explicit human confirmation
Secret masking	Connection strings, passwords, API keys redacted in tool outputs before LLM sees them
Path traversal protection	Terraform reader blocks access outside the repository root directory
IAM least-privilege	Lambda role has minimum required permissions (ECS read, CW read, CE read, Bedrock invoke)
Audit trail	All tool calls, inputs, outputs, and decisions logged to DynamoDB
Data residency	All Bedrock processing in-region (eu-west-3); no data leaves the AWS account

7 Deployment

The system supports two deployment modes:

7.1 Interactive CLI

Engineers run `python -m agent.main interactive` locally, using their AWS credentials. The Rich terminal UI renders Markdown responses, tables, and sparkline charts for real-time monitoring.

7.2 Serverless (Lambda + EventBridge)

For automated governance, the agent is deployed as an AWS Lambda function (Python 3.12, 512 MB, 15-minute timeout) triggered by EventBridge schedules:

- **Daily health check:** 7:00 UTC — generates comprehensive health report, runs anomaly detection, archives to S3, posts summary to MS Teams.
- **Weekly drift scan:** Compares Terraform desired state vs live infrastructure across all environments.
- **Monthly FinOps review:** Cost analysis with rightsizing recommendations.

8 Evaluation

8.1 Production Deployment

The agent was deployed in production managing infrastructure across the configuration shown in Table 3.

Table 3: Production deployment scope

Environment	Region	ECS Services	Role
Development	eu-west-3	9	Development & testing
Pre-production	eu-west-3	9	Staging & validation
Production EU	eu-west-3	9	Primary (read/write)
Production US	us-east-1	9	Secondary (read-only)

8.2 Operational Metrics

Table 4: Agent operational metrics

Metric	Value
Registered tools	53
Operational domains	10
Max iterations per query	30
Average tools invoked per query	5–12
Decision latency (end-to-end)	15–45 seconds
Lambda errors	0
Monthly operating cost	$7–23
Environments managed	4
AWS regions	2
ECS services monitored	9+ per environment

8.3 Cost-Benefit Analysis

Table 5: Monthly operating cost breakdown

Component	Estimated Monthly Cost
Bedrock inference (scheduled + interactive)	$5–15
Lambda runtime + EventBridge schedules	$0.50–2.00
CloudWatch / Logs Insights / Cost Explorer API	$1–5
S3 report archive + DynamoDB tables	$0.20–1.00
Total	**$7–23**

The agent's first cost optimization analysis typically identifies savings that exceed multiple years of its own operating cost. In our production deployment, the initial FinOps scan identified over-provisioned services with potential savings of $200+/month—paying for 8–28 years of agent runtime.

8.4 Investigation Efficiency

We compared the time required for common operational tasks with and without the agent (Table 6).

Table 6: Task completion time comparison

Task	Manual	Agent	Reduction
Cross-env cost optimization analysis	2–4 hours	2 min	98%
Drift detection (all environments)	1–2 hours	45 sec	99%
Incident root cause investigation	30–90 min	3 min	95%
Environment parity comparison	45–60 min	30 sec	99%
Executive health report generation	1–2 hours	1 min	98%

9 Discussion

9.1 Advantages Over Traditional Approaches

The agent's key advantage is *compositional reasoning*: it can combine information from multiple sources in ways that would require significant manual effort. A single query like "What happened to the charger service in the last 24 hours?" triggers a coordinated investigation across deployment history, logs, metrics, queue health, and configuration state.

9.2 Limitations

- **LLM reasoning boundaries:** Complex multi-step reasoning occasionally produces incomplete analyses, mitigated by the playbook system which provides proven investigation templates.
- **Latency:** End-to-end response time (15–45 seconds) is acceptable for operational queries but not for real-time alerting.
- **Model dependency:** The system depends on Amazon Nova Pro availability; model changes may require prompt tuning.

9.3 Future Work

- **Guardrailed automation:** Expand approved low-risk automated remediations.
- **Policy-as-code integration:** Enforce compliance checks before PR creation.
- **Multi-team adoption:** Slack/Teams bot for team-wide access with shared playbooks.
- **Executive scorecards:** Monthly tracking of time saved, prevented incidents, and captured savings.

10 Conclusion

We presented the Infrastructure Intelligence Agent, an AI-powered system that transforms cloud infrastructure management from fragmented, manual tool-hopping into a unified natural-language interface. Key achievements include:

1. **53 tools across 10 domains** providing comprehensive infrastructure visibility.
2. **Adaptive learning system** enabling continuous improvement through auto-extracted insights, baselines, feedback, and playbooks.
3. **Security-first architecture** with read-only defaults, approval gates, and complete audit trail.
4. **Production deployment** managing 4 environments across 2 regions at $7–23/month with zero errors.

5. **95–99% reduction** in investigation time for common operational tasks.

The system demonstrates that LLM-based agents with domain-specific tools and adaptive learning can meaningfully transform cloud operations from reactive firefighting to proactive, data-driven infrastructure governance.

Acknowledgments

The author thanks the Cloud Platform team for supporting this research and providing access to production infrastructure.

References

[1] Y. Dang, Q. Lin, and P. Huang, "AIOps: Real-World Challenges and Research Innovations," *Proc. ICSE-SEIP*, pp. 4–13, 2019.

[2] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning," *Proc. ACM CCS*, pp. 1285–1298, 2017.

[3] P. Wang et al., "CloudRCA: A Root Cause Analysis Framework for Cloud Computing," *IEEE Trans. Services Comput.*, vol. 16, no. 4, pp. 2584–2597, 2023.

[4] T. Schick et al., "Toolformer: Language Models Can Teach Themselves to Use Tools," *Proc. NeurIPS*, 2023.

[5] S. G. Patil et al., "Gorilla: Large Language Model Connected with Massive APIs," *arXiv:2305.15334*, 2023.

[6] T. Richards, "Auto-GPT: An Autonomous GPT-4 Experiment," GitHub Repository, 2023. Available: https://github.com/Significant-Gravitas/AutoGPT

[7] H. Chase, "LangChain: Building Applications with LLMs," 2023. Available: https://langchain.com

[8] HashiCorp, "Terraform: Infrastructure as Code," 2023. Available: https://www.terraform.io

[9] Amazon Web Services, "Amazon Bedrock — Generative AI Service," 2024. Available: https://aws.amazon.com/bedrock/

[10] Amazon Web Services, "Bedrock Converse API — Tool Use," 2024. Available: https://docs.aws.amazon.com/bedrock/

Manuscript received: April 2026