

# From Reactive Prompts to Autonomous Agents: LLMs for Multi-Cloud Infrastructure

By : Praneeth Kamalaksha Patil [Equinix IncConf42.com](https://EquinixIncConf42.com) Prompt

Engineering 2025



# The Multi-Cloud Challenge

## Today's Reality

Enterprises operate across AWS, Azure, GCP, and hybrid environments simultaneously. Managing configurations, policies, and security across these platforms creates exponential complexity.

Traditional tools struggle with cross-provider orchestration, leading to manual reconciliation, configuration drift, and increased operational overhead.



# Current State: Reactive LLM Usage

## Template Generation

Teams use ChatGPT or Copilot to generate Terraform configurations or Kubernetes manifests on demand.

## One-Off Scripts

Quick automation scripts created reactively when issues arise, without continuous validation.

## Documentation Queries

Looking up syntax and best practises for specific cloud services across providers.

Whilst useful, these approaches only scratch the surface of what's possible with LLMs in infrastructure management.



# The Vision: Autonomous Infrastructure Agents

1

## Continuous Awareness

Real-time state monitoring across all providers

2

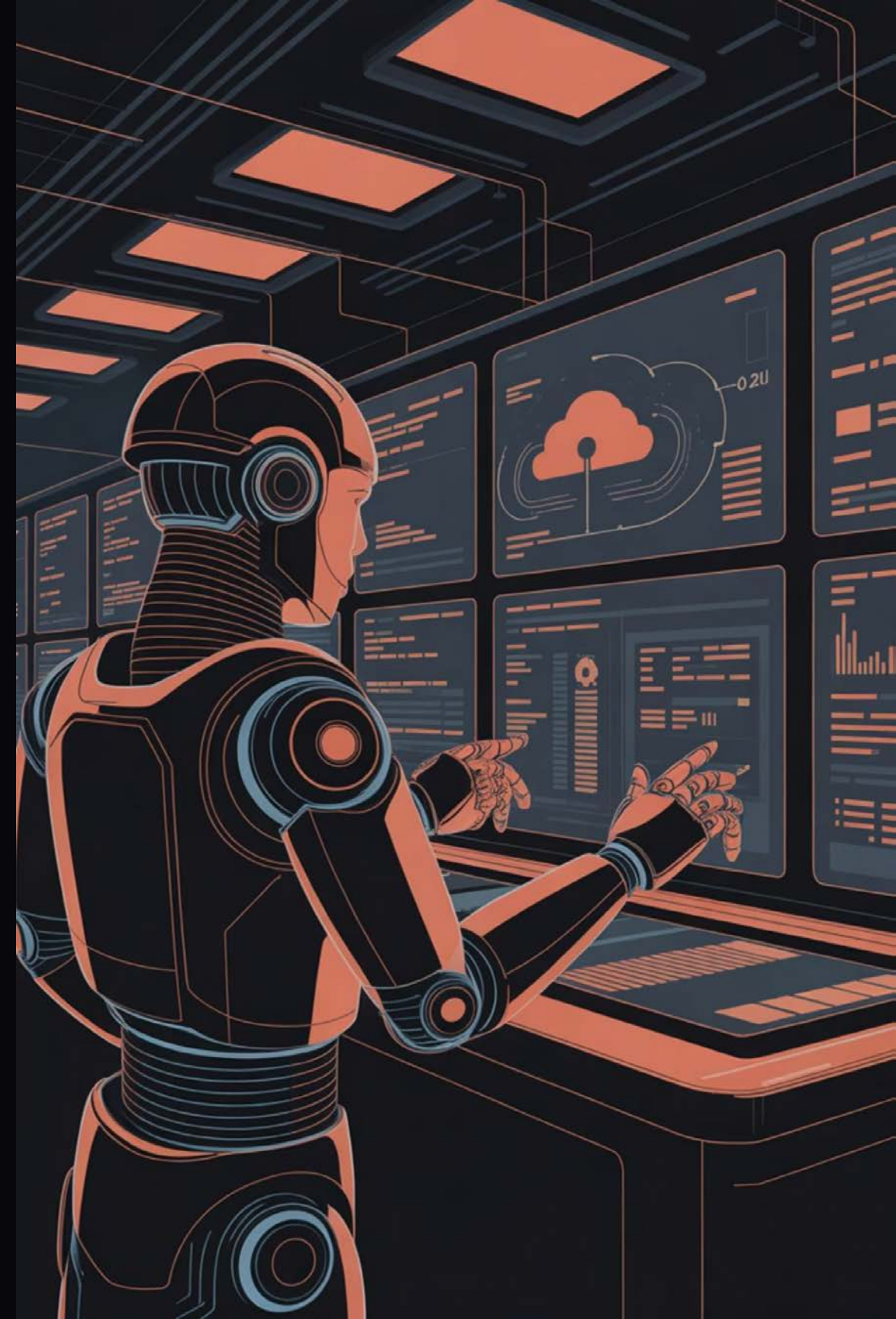
## Proactive Detection

Identifying drift and inconsistencies before impact

3

## Autonomous Resolution

Self-correcting configurations with human oversight



# Core Architecture: RAG for Infrastructure

## Retrieval-Augmented Generation

RAG enables LLMs to access context beyond single prompts. For infrastructure, this means querying:

- Current state files from Terraform/CloudFormation
- Historical deployment logs and configurations
- Security policies and compliance requirements
- Provider documentation and API specifications

The model "sees" the entire environment, not just isolated snippets.



# Multi-Stage Verification Pipeline

0

## <sup>1</sup>Syntax Validation

Check for correct HCL, YAML, or JSON structure before any deployment attempt.

0

## <sup>3</sup>Policy Compliance

Ensure configurations meet organisational security, cost, and governance policies.

0

## <sup>2</sup>Semantic Analysis

Verify logical consistency - do security groups actually allow the required traffic?

0

## <sup>4</sup>Dry-Run Testing

Execute terraform plan or equivalent to preview changes before applying.

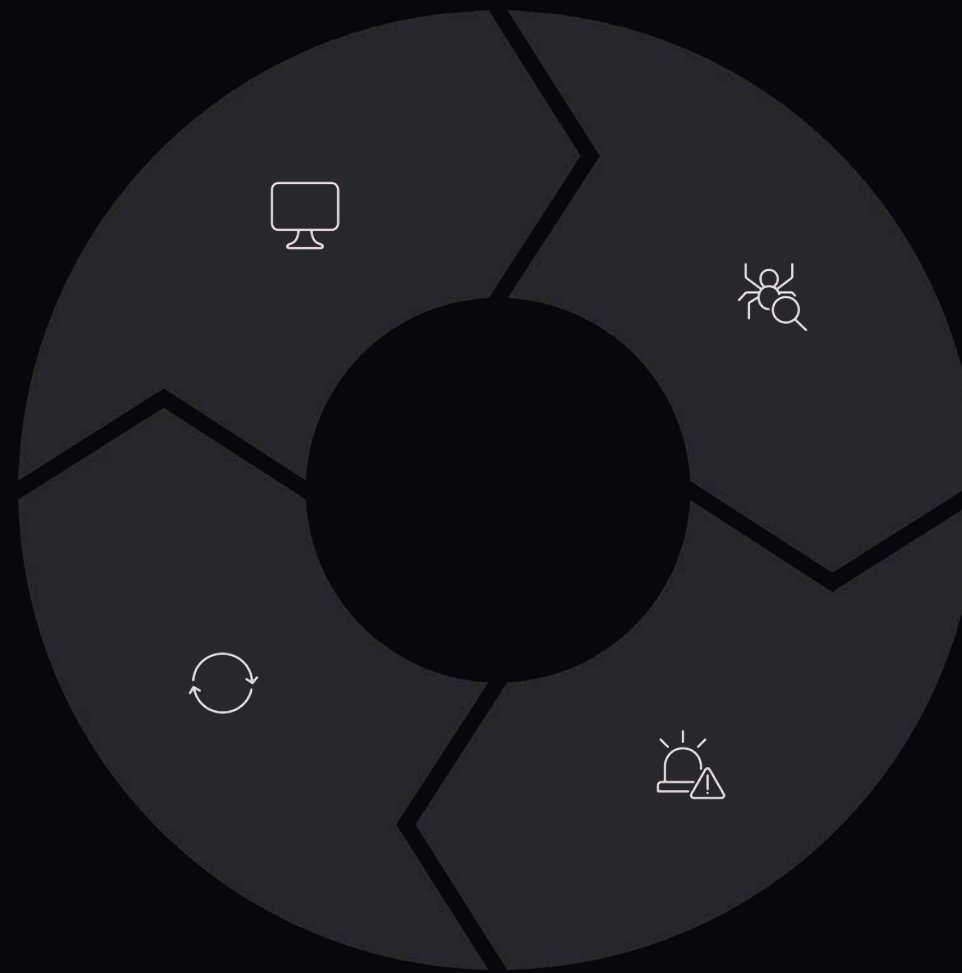
# Continuous Infrastructure Awareness

**Monitor State**  
Track configurations across all cloud providers in real time

**Reconcile**  
Propose or execute corrective actions with approval workflows

**Detect Drift**  
Identify deviations from desired state automatically

**Alert Teams**  
Notify relevant stakeholders with contextual information



# Practical Implementation Example

## Scenario

A security group rule was manually modified in AWS, creating drift from the Terraform state.

The LLM agent detects this within minutes.

It retrieves the intended configuration, compares current state, and generates a corrective pull request with explanation and impact analysis.

```
# Detected drift in security
groupaws_security_group.app_sg - ingress rule modified #
Agent action:1. Retrieved Terraform state2. Compared with
AWS API response 3. Generated corrective config4. Created
PR with context5. Notified security team
```



An abstract graphic on the left side of the slide. It features a central shield with a red-to-orange gradient, outlined in white and orange. The shield is set against a dark background with orange and black geometric shapes radiating from behind it. Two rectangular frames, one on each side of the shield, contain orange warning triangles with exclamation marks. At the bottom, there are concentric orange circles and lines suggesting a floor or a base.

# Critical Risks to Address

1

## Hallucinated Configurations

LLMs can confidently generate syntactically correct but semantically broken infrastructure code. Always validate through multiple stages.

2

## Security Exposure

Over-privileged LLM access can leak credentials or create vulnerabilities. Scope permissions strictly and rotate regularly.

3

## Cost Escalation

Large-scale inference across infrastructure states can become expensive. Implement caching and selective querying strategies.

# Mitigation Strategies

## Validation Gates

Implement automated testing at every stage - syntax, policy, security, and cost analysis before deployment.

## Least Privilege Access

Grant LLM agents only the minimum permissions required for their specific tasks with time-limited credentials.

## Cost Controls

Monitor inference costs, cache frequent queries, and use smaller models for simple validation tasks.

# Building Trust in LLM-Augmented Operations

## Transparency

Log all LLM decisions and reasoning



## Human Oversight

Require approval for critical changes



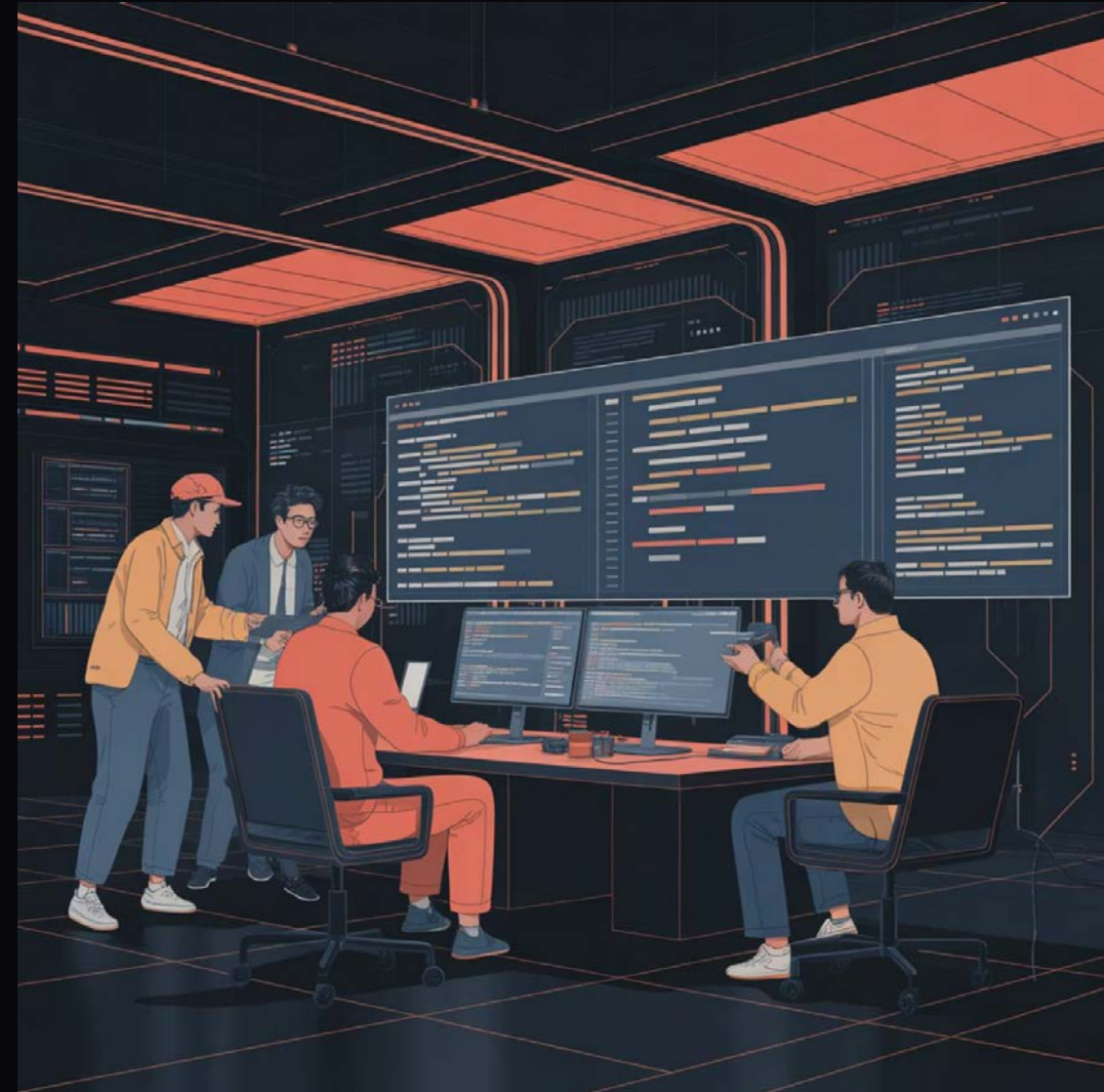
## Rollback Capability

Maintain instant rollback mechanisms



## Continuous Learning

Refine models based on outcomes



# Framework for Implementation




## Phase 1: Reactive Enhancement

Improve existing prompt-based workflows with better context and validation



## Phase 2: RAG Integration

Build retrieval systems that give LLMs access to infrastructure state and documentation



## Phase 3: Continuous Monitoring

Deploy agents that maintain awareness and detect drift automatically



## Phase 4: Autonomous Actions

Enable self-healing with appropriate guardrails and approval workflows

# Key Takeaways



## Evolution Beyond Prompts

Move from reactive code generation to continuous autonomous infrastructure management



## Robust Architecture

Combine RAG, multi-stage verification, and orchestration for reliable operations



## Risk Management

Address hallucinations, security, and costs through careful design and validation



## Incremental Adoption

Start with enhancements to existing workflows, build towards autonomy progressively



Thank

You!