International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Intent-Driven Fleets: An Agentic AI Framework for Cloud Elasticity

Somdutt Brajaraj Patnaik

Abstract: The evolution of cloud infrastructure toward hyper-scale deployments has exposed the fundamental inadequacy of reactive, threshold-based auto-scaling mechanisms. As digital services grow to serve global user bases during concentrated seasonal demand windows, the gap between high-level business objectives and low-level infrastructure execution has widened into a structural operational failure. Intent-Driven Fleets (IDF) address this gap through an autonomous orchestration framework that coordinates specialized AI agents, Commander, Forecasting, Provisioner, and Efficiency via the Model Context Protocol (MCP), enabling infrastructure to reason about goals rather than execute pre-written rules. The framework proposes the Plan-Execute-Observe-Reflect (PEOR) cycle, a formalized iterative process in which infrastructure anticipates demand, breaks business intent down into dependency-based execution plans, accepts business-layer telemetry to provide a context in which decisions are made, and continually optimizes provisioning behaviour via long-term memory. Security is also achieved by a deterministic guardrail layer, which is directly implemented as part of the MCP server that ensures that agent actions are not unlimited financially by permitting only signed authorization tokens, which are verified prior to each tool call. Individually identified engineering issues of this architecture are: context window congestion when receiving full telemetry, tool-call latency buildup amidst multi-region provisioning sequences, and concurrency conflicts necessitating distributed intent locking. The IDF framework establishes intent as the most effective abstraction for managing hyper-scale cloud environments, pointing toward a genuinely autonomous operational paradigm where global infrastructure responds directly to business goals without requiring continuous human translation at every execution step.

Keywords: *Cloud Elasticity, Multi-Agent Systems, Intent-Driven Orchestration, Model Context Protocol, Autonomous Infrastructure*

Introduction

The architecture of modern cloud infrastructure is under serious pressure. As digital services expand to serve hundreds of millions of users, particularly during concentrated seasonal demand windows, the foundational assumptions behind traditional scaling models are beginning to fracture [1]. Reactive, threshold-based auto-scaling was designed for a world where traffic grew incrementally and predictably. That world no longer exists.

At the heart of this problem is what can be described as the "Translation Gap." A business executive defines an objective, such as scaling to address a market of 200 million users during peak seasons, which must then be manually translated by engineering teams into thousands of lines of configuration files, infrastructure-as-code definitions, and imperative scaling policies [2]. This manual translation is slow, error-prone, and structurally disconnected from the original business intent.

Today's hyper-scale environments demand a fundamentally different approach, one where infrastructure does not wait to be told it is overwhelmed, but anticipates demand before it arrives [3]. This shift requires moving away from imperative rule-sets authored by human engineers and toward systems capable of reasoning about goals, decomposing them into actionable infrastructure tasks, and continuously reflecting on their own performance outcomes. The Intent-Driven Fleet (IDF) framework, coordinated through a Multi-Agent System (MAS) and the

Independent Researcher, USA

Model Context Protocol (MCP), represents precisely this shift, a move from "Goal-Defining" to "Goal-Seeking" infrastructure orchestration [4]. This article examines the architectural principles, technical mechanisms, operational outcomes, and honest limitations of this agentic approach to cloud elasticity.

1. The Limitations of Reactive Scaling

Traditional elasticity models are fundamentally reactive. Mechanisms like Kubernetes Horizontal Pod Autoscaling (HPA) rely on resource-based telemetry, primarily CPU and memory utilization, to trigger provisioning decisions [5]. At the scale of 200 million users, this reactive posture introduces three interlocking failure modes that collectively constrain what is achievable.

The first is the "Provisioning Lag." The time required to request, initialize, and bootstrap new compute nodes frequently exceeds 300 seconds. In a high-velocity traffic surge, this lag causes what is aptly termed a "Death Spiral": existing nodes become overwhelmed, latency spikes, and connection retries from clients further congest the network, all before new capacity comes online [2]. The system is attempting to recover from a condition that its own design made inevitable.

The second failure mode is the absence of cross-layer contextual awareness. Infrastructure is not simply compute; it is a complex web of stateful services, load balancers, and network ingress points. Imperative scaling rules are typically siloed. A CPU-based trigger for a web tier has no understanding that the underlying database clusters require a shard-split or a warm-up phase 30 minutes before traffic arrives [6]. This lack of "Intent-Awareness" means that while compute may scale correctly, the stateful backplane remains a bottleneck, leading to total system collapse despite thousands of idle CPU cores sitting available.

The third failure mode is the cognitive impossibility of manually managing multi-cloud seasonality. Engineers are forced to over-provision as a safety buffer, generating massive financial waste. In a multi-cloud environment spanning 12 or more global regions, a human-centric operations team cannot simultaneously optimize for real-time spot-instance pricing, regional latency variations, and capacity availability [5]. The cognitive load required to manage a 200-million-user event has categorically exceeded human capacity, necessitating a shift to an agentic model [3].

Failure Mode	Root Cause	Impact
Provisioning Lag	Bootstrap time exceeds surge velocity	Death Spiral latency spike before capacity is online
Siloed Intelligence	Scaling rules isolated per service tier	Stateful backplane becomes a bottleneck despite idle compute
Multi-Cloud Cognitive Overload	Human capacity exceeded across regions	Over-provisioning waste and missed optimization opportunities

Table 1: Limitations of Reactive Scaling [3, 5, 6]

2. The Intent-Driven Architecture

The IDF architecture consists of two synchronized layers: a reasoning layer that consists of dedicated AI agents and an execution layer that is an MCP Server. Put collectively, the layers close the divide between a business purpose on the high-level and the infrastructure activities at the low-level needed to achieve this purpose [4].

The reasoning layer is comprised of four specialized agents that have a specific responsibility in their operation. The Commander Agent is the highest-level organizer. It takes the business intent and decomposes the tasks recursively, dividing the high-level goal into a dependency-ordered graph of infrastructure sub-tasks and assigning them to downstream agents [7]. The **Forecasting Agent** is responsible for demand

intelligence. It continuously queries telemetry via the MCP server to predict the trajectory or "slope" of user arrival, allowing the system to act ahead of demand rather than in response to it [3].

The **Provisioning Agent** is the execution arm. It translates decomposed sub-goals into concrete infrastructure actions by invoking tools from the MCP Tool Library, calling operations such as warm_load_balancer() and apply_terraform_plan() to materialize the intent as running infrastructure [1]. The **Efficiency Agent** operates in the post-peak reflection phase, reviewing execution logs, identifying over-provisioned capacity, and writing optimization recommendations into the system's long-term memory for use in future cycles [7].

The **MCP Server** underpins all agent interactions. It provides the standardized toolset through which agents act on infrastructure, and it enforces the deterministic guardrail layer that validates every tool call before execution. The MCP server is simultaneously the control plane, the tool registry, and the compliance boundary of the entire system [4], [8].

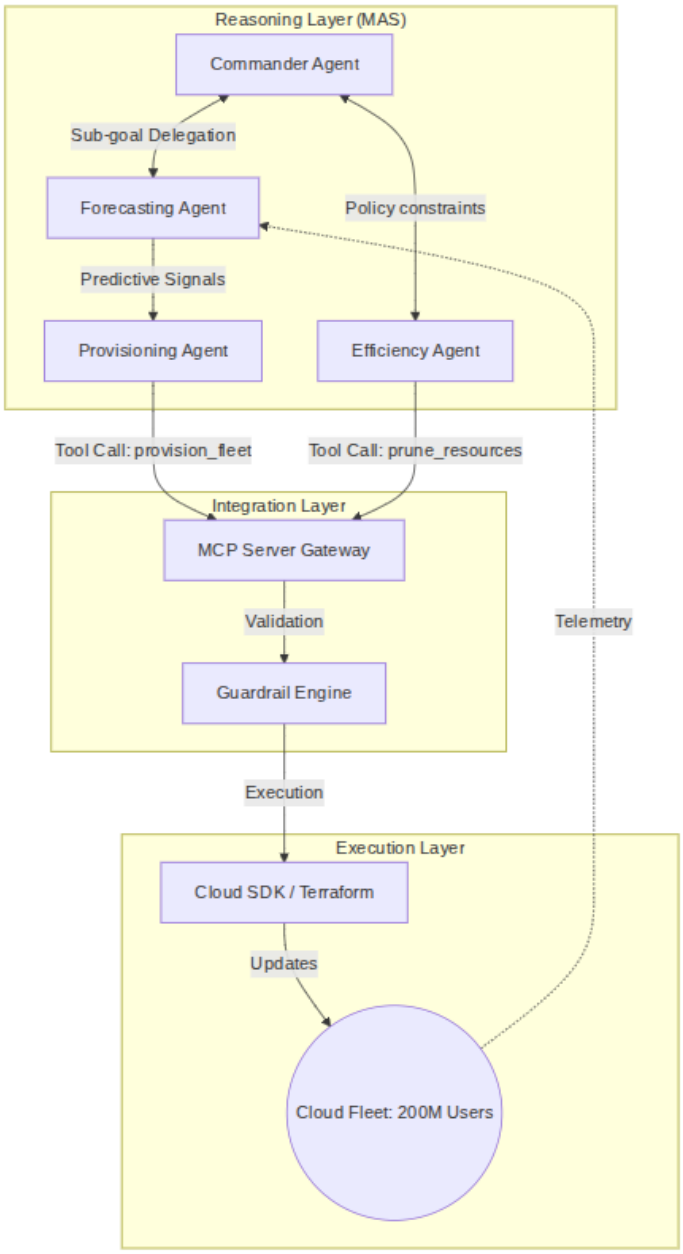

Fig. 1: IDF architecture

Agent	Primary Role	Key Output
Commander Agent	Recursive task decomposition	Dependency graph of ordered infrastructure actions
Forecasting Agent	Demand trajectory prediction	Predictive telemetry signals for pre-emptive scaling
Provisioning Agent	Infrastructure execution	Tool calls to IaC, load balancers, and node groups
Efficiency Agent	Post-peak reflection	Long-term memory updates and optimization recommendations
MCP Server	Tool registry and compliance boundary	Guardrail-validated execution of all agent tool calls

Table 2: IDF Agent Roles and Responsibilities [1, 3, 4]

3. The Plan-Execute-Observe-Reflect (PEOR) Cycle

The IDF replaces static scripts with iterative reasoning through a formalized four-phase cycle. The system is framed as an optimization problem: given a Business Intent (I), the system must identify a configuration (C) that satisfies the objective of minimizing a combined cost and latency function across 200 million users [2]. The PEOR cycle is the mechanism by which this optimization is continuously pursued.

3.1 Planning: Recursive Task Decomposition

The planning phase begins when the Commander Agent parses the business intent and identifies that a 200-million-user peak requires infrastructure actions that must occur in a specific sequence. The Commander generates a "Dependency Graph" that captures these ordering constraints [7]. In the case of a large seasonal peak, the graph encodes that a global database shard-split must be completed before compute scaling begins because scaling compute into an unprepared database tier simply relocates the bottleneck rather than removing it [6].

3.2 Execution: Predictive Warming and Mixed-Fleet Provisioning

The Provisioning Agent selects tools from the MCP Tool Library and executes them in the sequence prescribed by the dependency graph. Two behaviors distinguish this execution from reactive scaling. First, Predictive Warming starts up load balancers, groups of nodes, and connection pools in advance of the traffic that will require them, removing the lag of providing infrastructure as a failure mode [5]. Second, a Mixed-Fleet Strategy is a method that integrates spot and on-demand cases, independently operating real-time spot pricing and availability across regions to find an equilibrium between high cost efficiency and the availability guarantee [9].

A prototype implementation of this logic is captured in the provision_seasonal_fleet tool. The tool accepts a target capacity, a list of deployment regions, and an intent identifier. It applies guardrail checks for high-scale provisioning events, pre-warms load balancers per region to an expected rate of 2,000,000 requests per second, and creates managed node groups using a mixed instance strategy [4]. The tool returns a structured per-region status report confirming provisioning has been initiated:

```
import mcp_sdk as mcp
from cloud_provider import aws_eks_client
from typing import Dict, List

@mcp.tool(
    name="provision_seasonal_fleet",
    description="Scales global clusters to handle 200M users."
```

```
)
async def provision_seasonal_fleet(
    intent_id: str, target_capacity: int, regions: List[str]
) -> Dict:
    results = {}
    if target_capacity > 500000:
        audit_log(intent_id, "High-Scale Provisioning Triggered")

    for region in regions:
        await aws_eks_client.pre_warm_lb(region=region, expected_rps=2000000)
        fleet_response = await aws_eks_client.create_managed_node_group(
            cluster_name=f"fleet-{region}",
            instance_types=["m5.2xlarge"],
            capacity_type="SPOT",
            min_size=target_capacity // len(regions)
        )
        results[region] = {"status": "Provisioning", "nodes": fleet_response['node_count']}
    return {"intent_status": "Executing", "regional_deployment": results}
```

3.3 Observation: High-Density Contextual Telemetry

The Forecasting Agent ingests "High-Density Context" from the MCP Server, not just infrastructure metrics, but business-level telemetry such as checkout success rates, session establishment latency, and queue depth trends [3]. This richer signal set enables the agent to distinguish between a legitimate seasonal demand surge and an anomalous event such as a DDoS attack. Both may produce identical spikes in connection volume at the infrastructure layer, but their signatures diverge sharply in business-layer telemetry, a distinction that determines whether the correct response is to provision more capacity or to activate traffic filtering [10].

3.4 Reflection: Long-Term Memory and Continuous Improvement

After a peak event concludes, the Efficiency Agent reviews post-peak logs and quantifies resource utilization efficiency. If the analysis reveals that 20% of provisioned compute in a region remained idle throughout the event, the agent encodes a recommendation into long-term system memory, for example, "Recommend a 15-minute earlier warm-up for next year to optimize instance limits in the affected region" [7]. This forms a self-enhancing feedback cycle: every peak incident directly feeds back to the provisioning response of the system to the next, with a diminishing difference between available capacity and real demand [9].

Phase	Agent Responsible	Mechanism	Outcome
Plan	Commander Agent	Dependency graph construction	Ordered infrastructure task sequence
Execute	Provisioning Agent	Predictive warming, mixed-fleet provisioning	Infrastructure is ready before demand arrives
Observe	Forecasting Agent	Business-layer telemetry ingestion	Demand vs. anomaly classification
Reflect	Efficiency Agent	Post-peak log review and memory update	Refined provisioning behavior for future cycles

Table 3: PEOR Cycle Phases [5, 6, 7, 9]

4. Security, Guardrails, and Financial Compliance

The autonomy that makes an intent-driven system powerful is also the source of its most significant operational risk. An agent acting within its defined authority can, through a sequence of individually valid actions, produce a collectively catastrophic or financially unbounded outcome. This condition, "Agentic Runaway," is not a software bug in the traditional sense. It is a goal-seeking behavior that optimizes correctly for a specified sub-goal while violating broader operational constraints that were never formally encoded [8].

Preventing this requires a guardrail layer that is deterministic, not probabilistic. The MCP server implements exactly this: every tool call issued by any agent is validated against a signed Budget JWT before execution is permitted. This token encodes the full authorization envelope, the maximum permissible spend rate, the approved regional scope, and the permitted instance classes for the current operational intent [4]. A tool called either falls within this envelope, or it does not. There is no reasoning path by which an agent can justify an exception.

This design is deliberate. A probabilistic or model-based guardrail, one that evaluates whether a given action "seems" compliant, introduces the same contextual flexibility that the guardrail is meant to constrain [11]. Determinism is the point. The audit log records every high-scale provisioning event and every guardrail validation, creating a compliance trail that supports post-event financial reconciliation and operational review [8]. The Guardrail Engine is not a feature of the system; it is a precondition for deploying the system responsibly.

Mechanism	Type	Purpose
Budget JWT	Signed authorization token	Enforces financial and regional scope limits per intent
Guardrail Engine	Deterministic validator	Approves or blocks every tool call before execution
Audit Log	Compliance record	Tracks all high-scale provisioning events for post-event review
Agentic Runaway Prevention	Constraint layer	Prevents valid sub-goal actions from violating broader constraints

Table 4: Guardrail and Security Mechanisms [4, 8, 11]

5. Scaling Challenges of the MCP Control Plane

The MCP server provides the first standardized control plane for AI agents to interact with physical infrastructure. However, operating this control plane at the scale of a 200-million-user event introduces three distinct engineering challenges that are not yet fully resolved [4].

The first is **Context Congestion**. The shared telemetry context made available to reasoning agents expands dramatically during a peak event, error logs multiply, regional status updates arrive in rapid succession, and the number of concurrent active tool calls increases sharply. This context can grow to exceed the effective token limits of the underlying language model, degrading agent reasoning quality precisely when the quality of that reasoning is most consequential [12].

The second is **Tool-Call Latency**. Each tool call passes through the guardrail validation layer before execution. For individual calls, this overhead is negligible. During coordinated multi-region provisioning sequences, the cumulative latency of sequential validations introduces measurable "Agentic Lag" in regional clusters, a delay in the execution pipeline that partially offsets the lead-time advantage that predictive warming is designed to provide [9].

The third is **Concurrency and Distributed Intent Locking**. When multiple agents issue write operations targeting shared infrastructure state simultaneously, for instance, two agents concurrently attempting to modify the node count of the same cluster, conflicting operations can produce inconsistent infrastructure state [6]. Addressing this requires Distributed Intent Locking at the MCP server level: a protocol that serializes conflicting write operations and ensures that agent actions compose coherently under concurrent execution. Designing a locking protocol that is itself not a throughput bottleneck at peak scale remains an open engineering problem [11].

Challenge	Description	Consequence
Context Congestion	Telemetry volume exceeds model token limits at peak load	Degraded agent reasoning quality during critical events
Tool-Call Latency	Cumulative guardrail validation delays across multi-region calls	Agentic lag that offsets predictive warming advantage
Distributed Intent Locking	Concurrent agent writes conflicts on the shared infrastructure state	Inconsistent infrastructure state without a serialization protocol

Table 5: MCP Control Plane Scaling Challenges [11, 12]

Conclusion

The transition from imperative, resource-oriented scaling to intent-driven fleet orchestration represents a categorical shift in how large-scale cloud environments are designed and managed. Traditional reactive mechanisms constrained by provisioning lag, siloed scaling logic, and the cognitive limits of human operations teams are architecturally incapable of meeting the demands of hyper-scale seasonal peaks. These bottlenecks are solved by the IDF framework that grounds all infrastructure behavior on business-level intent, synchronizes stateful and stateless infrastructure layers with one another through recursive task decomposition, and allows continuous operational improvement by agent-driven reflection and long-term memory. Financial compliance and security are ensured by a deterministic guardrail layer that enacts authorization limits on all agent-initiated activity so that autonomous goal-seeking behaviour operates within approved operational envelopes. Although engineering issues are still especially relevant in the context of coping with congestion of contexts, minimizing agentic lag in validation pipelines, and introducing effective distributed locking mechanisms, these are solvable problems on top of an otherwise reasonable architectural base. The next step involves Future directions toward Adversarial Chaos Agents, which can be used to check the resilience of systems to unpredictable infrastructure failures, and additionally harden autonomous deployments until full-scale adoption of production. The evidence collectively supports a clear conclusion: intent is the most powerful and practical abstraction available for cloud management at scale, and the path toward a genuinely autonomous, no-human-intervention operational paradigm is no longer theoretical; it is an available architectural pattern whose adoption is now a matter of organizational readiness.

References

[1] Ahmed Barnawi et al., "The views, measurements and challenges of elasticity in the cloud: A review," Computer Communications, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0140366419319516

[2] AWS, "Unlock new value with agentic AI on AWS". [Online]. Available: https://aws.amazon.com/ai/agentic-ai/?trk=624522eb-7b74-4623-b50b-12a12eff8837&sc_channel=ps&ef_id=Cj0KCQjw7IjOBhDyARIsAFzrWQz5JpMg7ygr-lbiV599sk7CJbLb22dXmKSXlzW7AS9t_177UPdawF0aAtdMEALw_wcB:G:s&s_kwcid=AL!4422!

3!795924513590!p!!g!!agentic%20ai!2352857246 9!196289121721&gad_campaignid=23528572469 &gbraid=0AAAADjHtp81WbJO_Wn5zyt2VgxzURdFs&gclid=Cj0KCQjw7IjOBhDyARIsAFzrWQz5JpMg7ygr-lbiV599sk7CJbLb22dXmKSXlzW7AS9t_177UPdawF0aAtdMEALw_wcB

[3] Yuxing Zhang et al., "Predictive Auto-Scaling in Distributed Cloud Environments Using Machine Learning," ICDIS '25: Proceedings of the 2nd International Symposium on Integrated Circuit Design and Integrated Systems, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3772326.3774726

[4] Model Context Protocol, "Architecture overview". [Online]. Available: https://modelcontextprotocol.io/docs/learn/architecture

[5] Rzadca, Krzysztof, et al. "Autopilot: workload autoscaling at Google." EuroSys '20: Proceedings of the Fifteenth European Conference on Computer Systems. 2020.[Online]. Available: https://dl.acm.org/doi/abs/10.1145/3342195.3387524

[6] Stefan Nastic et al., "A Serverless Real-Time Data Analytics Platform for Edge Computing," IEEE Internet Computing, 2017. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7994559

[7] IBM, "Multi-agent orchestration". [Online]. Available: https://www.ibm.com/products/watsonx-orchestrate/multi-agent-orchestration?utm_content=SRCWW&p1=Search&p4=2368123929001&p5=p&p9=187402692430&gclsrc=aw.ds&gad_source=1&gad_campaignid=22029767487&gbraid=0AAAAAD-_QsT1gyIkcksIsL9yClbGu20SB&gclid=Cj0KCQjw7IjOBhDyARIsAFzrWQx9ik1nw1gagLHvux02YxaIBX9GZQEYdwziu--96zlaGqmt9rLcqaMaAu_uEALw_wcB

[8] Perspicientia Consultancy Private Limited, "Designing Guardrails for Autonomous AI Systems," LinkedIn 2026. [Online]. Available: https://www.linkedin.com/pulse/designing-guardrails-autonomous-ai-systems-perspicientia-bk4gc/

[9] Benjamin Hindman et al. "Mesos: A platform for {Fine-Grained} resource sharing in the data center". [Online]. Available: https://www.usenix.org/legacy/event/nsdi11/tech/full_papers/Hindman.pdf

[10] Ghaith Dkmak et al., "AI-Driven Anomaly Detection in Cloud-Native Microservices: The Night's Watch Algorithm," Appl. Sci. 2025. [Online]. Available: https://www.mdpi.com/2076-3417/15/23/12762

[11] Uchi Uchibeke et al., "Before the Tool Call: Deterministic Pre-Action Authorization for Autonomous AI Agents", 2026. [Online]. Available: https://arxiv.org/pdf/2603.20953

[12] McKinsey, "What is a context window?", 2024. [Online]. Available: https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-a-context-window