Unified Control Plane Architecture for Multi-Project AI Agent Fleets

Executive Summary

The rapid transition from single-agent experimental workflows to enterprise-scale, multi-project agent fleets has exposed critical gaps in current infrastructure. Organisations attempting to manage hundreds of autonomous agents across competing projects face severe challenges in resource contention, financial accountability, tenant isolation, and operational visibility. To solve these challenges, this report proposes the adoption of an "Agent Operating System" (Agent OS)—a unified control plane that treats large language models (LLMs), tools, memory, and execution environments as addressable resources orchestrated by a centralised kernel.[1]

The proposed architecture acts as an abstraction layer between developer-facing application logic and underlying infrastructure. By decoupling the execution plane (where agents run) from the control plane (where policies, budgets, and schedules are governed), organisations can safely scale multi-tenant deployments. This paradigm prevents context fragmentation, eliminates scheduling unfairness, and enforces strict hardware-level environment isolation.[2] Furthermore, it embeds continuous financial discipline through real-time cost attribution, ensuring that autonomous actions remain within strict budgetary and legal constraints.[5] The resulting architecture allows a single operations team to oversee dozens of fleets without compromising security, quality, or human authority.

Terminology Glossary

To establish a precise operational vocabulary, the following entities and concepts are defined within the context of multi-project agent management.

Term	Definition
Organisation	The root tenant encompassing all workspaces, portfolios, and global mandatory policies.
Workspace	A logical partition within an organisation, often corresponding to a department or region.[6]
Portfolio	A collection of related programmes or

	projects grouped for financial and strategic oversight.
Programme	A coordinated group of projects managed together to achieve a broader business objective.
Project	A bounded initiative with a specific business objective, owner, budget, and data boundary.[5]
Fleet	A coordinated collection of agents operating under a shared configuration, purpose, and resource quota within a project.
Swarm	A highly cooperative, dynamically scaling subset of a fleet executing tightly coupled parallel tasks.
Workflow	A defined sequence of tasks executed by agents to achieve a reproducible outcome.
Agent Role	A reusable, abstract definition of an agent's capabilities, prompt instructions, and tool access (e.g., "Security Reviewer").
Named Agent	A persistent logical identity assigned to an agent role within a specific project, maintaining long-term memory and performance history.
Agent Instance	An ephemeral execution process actively processing a task on behalf of an agent role or named agent.
Worker	The underlying compute resource or container facilitating the execution of an agent instance.

Orchestrator	The system component responsible for managing the lifecycle, queuing, and scaling of workers.
Control Plane	The centralised governance system responsible for registries, policies, scheduling, budgets, and audits.
Data Plane	The pathways through which raw memory, vector embeddings, and context flow between agents and storage.
Execution Plane	The isolated runtime environments where agents invoke models, execute tools, and manipulate data.[2]
Policy Engine	A system (e.g., Open Policy Agent) that evaluates runtime requests against defined declarative rules.[7]
Scheduler	The kernel module that determines priority and allocates model/compute resources to queued tasks.[2]
Resource Pool	A shared aggregate of computational and LLM API capacities available for allocation.
Environment	A hardware-isolated sandbox (typically a microVM) where untrusted agent code and tool logic run.[3]
Tenant	An isolated entity (Organisation or Workspace) whose data, configurations, and users are strictly separated from others.[6]
Budget	A predefined financial limit capping the allowable spend for a project, fleet, or tag within a given period.[9]
Quota	A volumetric limit (e.g., tokens per minute,

	concurrent environments) restricting resource consumption.[6]
Cost Centre	The accounting entity to which a project's LLM, compute, and operational costs are attributed.[9]
Governance	The framework of policies, approvals, and ownership ensuring agents operate safely and legally.
Audit	The immutable record of all system state changes, configuration overrides, approvals, and execution traces.
Agent OS	A comprehensive system providing kernel-like primitives (scheduling, isolation, I/O mediation) for AI agents.[2]

The Agent Operating System Concept

The concept of an Agent Operating System extends the traditional OS metaphor to the orchestration of autonomous AI entities. Just as a conventional OS abstracts hardware resources (CPU, RAM, disk) away from application processes, an Agent OS abstracts models, context memory, and execution environments away from individual agents.[1]
This abstraction is vital because unrestricted access to LLM endpoints or unmediated tool access leads to severe resource contention, context fragmentation, and state corruption.[2] Without a kernel-level scheduler, a fast triage agent might be starved of resources by a slow research agent monopolising the model API.[2]

Conceptual Mapping

The following mapping illustrates how classical OS constructs translate to the multi-agent platform architecture:

Operating-System Concept	**Agent-Platform Equivalent**
Process	Agent run or task execution
Service	Persistent agent role or fleet service

User account	Agent identity or operator identity
Permission	Tool, data, environment, or action capability
Scheduler	Cross-project task scheduler (prioritising LLM calls) [2]
Process group	Fleet or workflow
Container	Isolated agent execution environment (e.g., Firecracker microVM) [3]
Filesystem	Project knowledge, artifact storage, and vector databases
Package version	Agent configuration version, prompt version, or tool dependency
System monitor	Fleet command centre and economic dashboard
Kill signal	Pause, cancel, or emergency stop via circuit breakers

Limitations of the Metaphor

While highly useful for system design, the OS analogy breaks down in certain areas. Conventional processes are deterministic; given the same inputs, a CPU yields the same outputs. Agent instances rely on probabilistic LLMs, meaning the Agent OS must account for hallucinations, retry storms, and semantic drift. Furthermore, traditional OS memory management deals with addressable bytes, whereas Agent OS context management deals with semantic truncation, summarisation, and vector retrieval.[2] The Agent OS must evaluate *quality* and *confidence*—metrics non-existent in classical process scheduling.

Organisational Hierarchy and Inheritance Model

A scalable hierarchy must support operations ranging from a single technical operator to a multi-national enterprise requiring strict tenant isolation and separate billing. The system models structural entities hierarchically to enforce inheritance of policies, budgets, and provider settings.[6]

Hierarchy Structure

The hierarchy scales down from the root tenant to the ephemeral task:

1. **Organisation:** The apex entity representing the enterprise. It holds global mandatory policies (e.g., GDPR compliance rules, global cloud provider bans) and the root budget.
2. **Workspace/Business Unit:** Logical divisions (e.g., EMEA, North America, Engineering, Marketing) acting as the primary multi-tenant boundary for internal teams. They maintain separate cost centres and administrative domains.[6]
3. **Portfolio/Programme:** Groupings of projects for financial forecasting and strategic alignment.
4. **Project:** The primary boundary for data isolation, containing specific business objectives, dedicated vector memory, and allocated budgets.[5]
5. **Fleet:** The operational unit of agents executing tasks toward the project's objective.
6. **Workflow & Task:** The granular units of execution processed by agents.
7. **Agent Instance:** The active execution thread handling a task.

Inheritance and Overrides

Inheritance flows downward. An Organisation may set a global policy restricting outbound network access to an approved list of domains. A Workspace inherits this policy. A Project owner may tighten this restriction (e.g., denying all outbound access except to an internal GitHub repository), but they cannot weaken the global rule.[12]
Configurations are layered: Organisation Defaults → Workspace Policy → Fleet-Template Defaults → Project Configuration → Fleet Configuration → Task-Level Overrides. Conflict resolution operates on the principle of maximum restriction: if policies contradict, the most restrictive rule prevails, and mandatory global policies cannot be overridden without explicit, audited emergency elevation.[12]

Registry Architecture

Authoritative registries form the backbone of the control plane, answering interrogations about platform state, ownership, and permissible actions.

Project Registry

The project registry serves as the system of record for all business initiatives. Valid project states include Draft, Provisioning, Active, Paused, Blocked, Degraded, Closing, Completed, Archived, Suspended, and Deleted. State transitions (e.g., Draft to Active) require budget owner approval.

Field	Description
Project ID	Unique cryptographic identifier for the

	project.
Status	Current operational state (e.g., Active, Archived).
Owners	Named Technical, Budget, and Security owners accountable for the project.
Associated Fleets	Array of Fleet IDs operating under this project's boundary.
Data Classification	Sensitivity tier (e.g., Public, Internal, Confidential, Restricted).
Approved Providers	Inherited or explicit list of permitted LLM APIs (e.g., Anthropic, Azure).
Cost Centre	Tag used by the API proxy for financial attribution.[9]

Example JSON Project Record:

```
JSON
{
 "project_id": "prj_9a8b7c6d",
 "name": "Market Intelligence Q3",
 "status": "Active",
 "data_classification": "Confidential",
 "owners": {
  "technical": "usr_112233",
  "budget": "usr_445566"
 },
 "fleets": ["flt_101", "flt_102"],
 "policies": {
  "approved_providers": ["openai", "anthropic"],
  "max_task_budget_gbp": 5.00
 },
 "cost_centre": "mktg_research_01",
 "archival_state": null
```

}

Fleet Registry

The fleet registry tracks active collections of agents. Fleets maintain states such as Registered, Provisioning, Ready, Active, Scaling, Degraded, Draining, Paused, Stopped, and Failed. A fleet transitioning to "Draining" ceases intake of new tasks while completing active workflows.

Field	Description
Fleet ID	Unique identifier linked to a parent Project ID.
Agent Roster	Catalogue of assigned Agent IDs and their authorized replica counts.
Template Version	Semantic version of the template used to spawn the fleet.
Resource Quotas	Hard limits on concurrent workers, token throughput, and queue depth.
Health Metrics	Real-time queue age, success rate, and error-rate circuit breaker status.

Agent, Provider, and Environment Registries

- **Agent Catalogue:** A library of reusable agent definitions containing default prompts, capability mappings, required tool permissions, context limits, and empirical evaluation histories.
- **Provider Registry:** A database of external LLM endpoints, managing changing price schedules, capability mappings, volume discounts, and rate limits.[6] Costs are calculated against the specific price version active at the exact time of execution.
- **Environment Registry:** Tracks the lifecycle of hardware-isolated sandboxes, including their Region, Runtime Image, Resource Limits (CPU/RAM), Network Policy configurations [12], and Expiry/Cleanup triggers.

Agent Lifecycle and Identity Model

The architecture enforces a strict conceptual separation between persistent logical identities, reusable configurations, and ephemeral execution processes.

Reusable Roles vs. Named Agents

- **Reusable Roles:** These act as configuration packages or blueprints (e.g., "Code Reviewer v2.1"). Deploying a role across multiple projects scales horizontally and ensures standardisation. The role possesses no memory; it is purely behavioral logic.
- **Named Agents:** These are persistent logical identities (e.g., "Security Reviewer Alpha") spawned from a role but bound to a specific project. Named agents build long-term performance histories and maintain project-specific vector memory. The system models a named agent as a specific deployment record linked to an isolated memory container. This structure mitigates the risk of configuration drift and prevents cross-project memory contamination, which occurs when a single instance tries to serve multiple disparate projects.[15]

Temporary vs. Persistent Agents

- **Temporary Agents:** Ephemeral entities created for a single task (e.g., a one-off document formatter). They possess ephemeral execution credentials, maintain no private long-term memory, and their compute environments are destroyed instantly upon task completion, minimising operational overhead and security risk.
- **Persistent Agents:** Entities with long-lived operational ownership, recurrent responsibilities, and persistent task queues (e.g., a Fleet Supervisor). It is crucial to note that a persistent agent *does not imply a persistent compute process*. The agent's identity and memory persist in a database, but its actual execution occurs within an ephemeral execution environment (sandbox) spawned strictly for the duration of a specific interaction.[16]

Shared Infrastructure vs. Shared Agents

Shared infrastructure refers to multi-tenant underlying systems (like a LiteLLM proxy or an E2B microVM cluster).[5] Shared agents are distinct logical entities (e.g., a Central Cost Optimiser) serving multiple projects. Shared agents must be designed with strict "pass-through" context; they receive task-scoped context for a single run and cannot persistently access all project memory indiscriminately.

Configuration System and Versioning

Configurations across the Agent OS must be stored as code in an immutable event store, ideally backed by Git semantics. Every execution must record the precise configuration versions utilised to guarantee auditability.[17]

Configuration categories include system instructions, prompts, tool definitions, fleet templates, permission policies, budget rules, and environment images. Versioning follows Semantic Versioning (SemVer) principles. When a fleet template is updated, the control plane assesses compatibility constraints—such as verifying that the new agent version supports the minimum required tool schema—before permitting deployment.

The configuration lifecycle proceeds through Draft, Test, Evaluate, Review, Approve, Publish,

Deploy, Monitor, Deprecate, and Retire phases. Automated regression evaluations run against A/B comparisons and canary deployments before a new agent prompt version is propagated widely, enabling immediate rollback upon detection of quality degradation.

Environment Isolation and Security Architecture

When deploying AI agents that generate untrusted code, execute shell commands, or browse the web, strict environment isolation is paramount. Current platforms relying on standard Docker containers or Kubernetes namespaces are insufficient for enterprise multi-tenancy due to shared host kernels and the risk of container escape vulnerabilities.[4]

Execution Plane Isolation (Sandboxing)

The Agent OS relies on hardware-enforced microVMs (e.g., Firecracker, utilised by E2B) for the execution plane.[3] Every agent instance is granted a dedicated kernel, filesystem, and network stack. This provides high-security, tenant-level isolation, ensuring that a compromised agent cannot access the orchestrator or bleed state into another project's sandbox.[4]

Network Policies and Egress Control

By default, the network policy for an execution environment denies all outbound internet traffic.[12] Outbound network access must be explicitly permitted via granular policies supporting domain-level (e.g., *.github.com) and CIDR block allowlists.[12] If a task requires downloading a dependency, the policy engine validates the request against the project's acceptable data classification constraints before injecting the allow-rule into the running sandbox.

Credential Scopes

The platform avoids long-lived, shared API keys. Instead, it employs Just-In-Time (JIT) access and Workload Identity. A secret broker issues short-lived, task-scoped credentials injected directly into the microVM's environment variables.[19] The credential expires the moment the task completes or the sandbox is terminated. Every sensitive action executed against an external API is thus cryptographically attributable to a specific task ID, Agent ID, and Project ID.

Memory Boundaries and Data Residency (GDPR)

Cross-project data leakage is prevented architecturally, not just procedurally. Projects are assigned separate vector indexes or distinct tenant collections; a shared agent cannot retrieve context across these hard boundaries.[15]

To comply with GDPR and EU Data Residency requirements, the control plane ensures that processing locations align with tenant policies. When agents use the Model Context Protocol (MCP) to interact with external systems (e.g., Salesforce, Notion), the Agent OS functions as a pass-through proxy. Tool payloads and personally identifiable information (PII) are processed in ephemeral memory and are never persisted at rest in the platform's logs or databases, enforcing a true zero-retention architecture.[17]

Scheduling and Quota Architecture

A defining characteristic of an Agent OS is its ability to schedule tasks globally while balancing competing cross-project priorities and resource quotas.[2]

Cross-Project Task Priorities

The scheduler resolves contention using a multi-level feedback queue combined with weighted fair queuing. Priority signals include business importance, deadline, incident severity, task age, and human approval status. The system prevents low-priority starvation by guaranteeing minimum resource floors for all active projects, while allowing critical incidents to preempt background generative tasks.

Task Pre-emption

Agent tasks are pre-emptible. If a high-priority incident occurs, the scheduler can pause a background agent at a safe checkpoint (e.g., between LLM calls or before writing to a database), suspend the microVM environment (snapping memory state to disk [16]), and return the compute resources to the pool. Irreversible tasks (e.g., executing a financial transaction or external communication) are locked and cannot be safely pre-empted once initiated.

Resource Quotas

Quotas govern concurrent tasks, token throughput, and compute time across the organisational hierarchy.[6]

Quota Type	Enforcement Mechanism	Exhaustion Behaviour
Hard Quota	Strict systemic cap (e.g., max concurrent VMs).	Task queued; alerts triggered.
Soft Quota	Threshold for capacity planning.	Task proceeds; warning logged.
Rate Limit	Provider-level API constraints (e.g., TPM).	Dynamic fallback to secondary provider.[5]
Budget Cap	Financial limit per time period.	Circuit breaker trips; human approval required.[5]

Economic Model and Budget Enforcement

Achieving financial discipline in a multi-provider ecosystem requires a provider-neutral

cost-accounting model. Operators cannot rely on delayed end-of-month cloud invoices; costs must be estimated, tracked, and attributed in real-time.

Budget Hierarchy and Tagging

The architecture utilises an LLM gateway/proxy (e.g., LiteLLM) to enforce budgets hierarchically across Organisations, Workspaces, Projects, and Teams.[6] Every model request includes metadata tags identifying the Project ID, Fleet ID, Cost Centre, and Environment.[5]
Budgets are configured with specific durations (e.g., daily, monthly) and limits.[9] Budget enforcement policies escalate dynamically:

1. **60% Utilisation:** Asynchronous alert to the project owner.[5]
2. **85% Utilisation:** Escalate to platform administrators; pause background tasks.
3. **100% Utilisation:** Hard cap enforced. Circuit breakers trip, disabling non-critical workflow paths. Re-enablement requires human budget-owner override.[5]

Unit Economics: Cost Per Completed Task

Simplistic metrics based solely on token prices obscure the true economic efficiency of an agent fleet. The Agent OS calculates the *fully loaded cost per accepted task*.
The economic formula incorporates variable and fixed costs, as well as failure probabilities: Expected Task Cost = Initial Execution Cost + (Probability of Failure × Expected Recovery/Rework Cost) + Shared Platform Overhead Allocation A cheap model that hallucinates frequently, requiring multiple automated retries and eventual human intervention, possesses a much higher fully loaded cost than an expensive, highly capable model that succeeds on the first attempt. Therefore, an output cannot be classified as an economic success unless it passes quality evaluation and is accepted by the business workflow.

Model-Routing Economics and Trade-offs

Model routing is determined dynamically based on requested capabilities (e.g., fast extraction, strong reasoning, vision) rather than hardcoded model names.
The router evaluates Quality-Cost-Latency trade-offs via a Pareto frontier:

- **Low-Risk Classification (Economy Class):** Prioritises cost and speed. Routes to the cheapest eligible model.[5]
- **Customer-Facing Urgent Response (Standard Class):** Balances quality and latency. Utilises mid-tier models with fast time-to-first-token.
- **Strategic Research/Security Review (Critical Class):** Prioritises completeness and evidence. Routes to frontier models with extended reasoning tokens, bypassing cost-optimisation constraints provided the project budget permits.

Governance, Policy-as-Code, and Approvals

Governance transitions from informal documentation to machine-enforceable rules managed by Policy Engines (e.g., Open Policy Agent).[7]

Autonomy Matrix

Agent actions are strictly classified by autonomy level, dictating the required governance overhead:

Level	Description	Example Action	Governance Requirement
0	Observe only	Read monitoring logs.	Standard RBAC.
1	Suggest	Propose a code refactor.	Output logging.
2	Reversible, low-risk	Create a draft PR.	Policy engine validation.
3	Broader, audited actions	Merge approved PR to staging.	Policy engine + automated audit.
4	High autonomy within bounds	Execute data pipeline.	Strict budget/quota guards.
5	Critical, irreversible actions	Modify production DB; external emails.	Multi-party human approval.

Approval Experience

For Level 5 actions, or actions exceeding forecasted budget thresholds, the control plane pauses execution and routes a request to the Approval Inbox. To prevent approval fatigue, the interface provides comprehensive context: the requesting agent, forecasted cost, data involved, proposed side effects, and reversibility. The human operator can approve the request, approve it with a tighter financial limit, reject it, or request an agent revision. Importantly, LLMs are never used as the final authority for policy enforcement or Level 5 approvals.

Auditability and Incident Management

The control plane maintains an append-only, tamper-evident audit log detailing who deployed an agent, which configuration version executed, the exact provider used, the data accessed, and any human overrides applied.

Kill Switches and Circuit Breakers

The Agent OS employs layered emergency controls.[5]

- **Automatic Circuit Breakers:** Monitor error rates, token velocity, and tool failures. If an agent loops and repeatedly fails a tool call, the circuit breaker opens, halting execution to prevent a retry storm and quota exhaustion. Breakers transition through Closed, Open, and Half-Open (testing recovery) states.
- **Kill Switches:** Manual or API-driven triggers that allow operators to enact scopes ranging from pausing a single task to terminating all external network access for an entire organisation.

Failure Scenarios

The system is designed to gracefully handle complex failures:

- *Scenario:* A shared agent retrieves memory from the wrong project.
  - *Prevention:* Tenant IDs are hard-enforced at the vector DB level; queries without matching project metadata are rejected by the data plane.[20]
- *Scenario:* A provider suffers latency degradation.
  - *Detection & Recovery:* The LiteLLM proxy detects timeout anomalies, trips the provider circuit breaker, and automatically reroutes traffic to a configured fallback provider without failing the parent task.[5]
- *Scenario:* A project approaches budget exhaustion.
  - *Containment:* At 85%, the scheduler automatically deprioritises background tasks and downgrades standard tasks to cheaper models, extending the remaining runway until human intervention.[5]

Dashboards and Interface Specification

The operator interface is designed for non-developer operators, utilising progressive disclosure to move from high-level business metrics to granular technical diagnostics.

Portfolio Overview Dashboard

The top-level view aggregates data across all workspaces:

Metric / Section	Operator View	Actionable Drill-Down
Project Health	45 Active / 3 Blocked	View blocked projects and failing tasks.
Financial Velocity	£4,500 spend (Forecast: £12,000)	Identify top 3 highest-spending fleets.
Provider Status	Provider A: Degraded	View automatic fallback

		routing status.
Approval Backlog	12 Urgent, 4 Standard	Navigate to Unified Approval Inbox.
Incident Alerts	1 Open Circuit Breaker	Inspect retry storm in Fleet 04.

Project and Fleet Command Centres

The project command centre allows operators to pause projects, adjust budgets, and alter provider allowlists. The fleet operations page drills down into agent rosters, workflow queues, and cost per accepted task. Operators can trigger emergency stops directly from the fleet view, suspending environments and preserving checkpoints instantly.

Archival and Retention Model

Agent projects have finite lifecycles. Leaving a completed project active exposes the platform to orphaned compute costs, stale credential abuse, and unnecessary GDPR retention risks.[22]

Archival Sequence

When a project owner triggers archiving, the control plane initiates a deterministic validation sequence:

1. Verify no tasks are actively running in the queue.
2. Terminate all active microVM execution environments.[3]
3. Revoke all task-scoped and project-scoped credentials from the secret broker.
4. Reconcile final API costs with the cost ledger.
5. Snapshot project metadata, fleet configurations, and audit records into cold storage.
6. Securely delete ephemeral memory caches and vector indexes to satisfy data minimization policies.[22]

Reactivation

If an archived project must be resumed, the system performs a reactivation assessment. It does not blindly resume execution. It verifies that old agent configurations are compatible with current system invariants, checks that historical providers are still authorized, and prompts operators to issue new credentials, ensuring outdated security assumptions are not reintroduced.

Build-Versus-Buy Analysis

Implementing this architecture requires integrating custom control plane logic with specialised infrastructure providers.

Component Category	Build (Custom)	Buy (Off-the-shelf / OSS)	Architectural Rationale
Registries & Dashboards	**Build**	-	Project registries require deep integration with specific enterprise business logic, cost centres, and custom UIs.
Agent OS Kernel	**Build** (Leverage OSS AIOS)	AIOS framework	Custom scheduling logic is required to balance multi-project priorities; academic AIOS concepts provide the structural foundation.[1]
Execution Sandboxes	-	**Buy** (E2B, Daytona)	Building secure Firecracker microVMs is highly complex. Rely on established providers for hardware-level isolation, sub-second boot times, and network egress control.[3]
API Proxy & Budgets	-	**Buy** (LiteLLM)	Existing proxies flawlessly handle multi-provider routing, multi-tenant tagging, fallback logic, and

			hierarchical budget caps out-of-the-box.[5]
Policy Engine	-	**Buy** (OPA)	Open Policy Agent (OPA) provides an industry-standard, well-tested engine for declarative policy-as-code evaluation.[7]
Vector Memory	-	**Buy** (Pinecone, Qdrant)	Use managed vector databases, but enforce strict multi-tenancy at the application logic layer via separate indexes.[20]

Phased Implementation Roadmap

Deploying a full Agent OS is a massive undertaking. The implementation is broken into seven progressive phases to deliver immediate operator value while building toward full autonomy.

Phase	Focus Area	Key Features Delivered	Operator Value
1	Registries & Visibility	Project/Fleet registries, Agent catalogue, basic dashboard, static routing.	Centralised visibility of all active initiatives and ownership.
2	Isolation & Security	MicroVM sandboxing (E2B) [23], network egress blocking [12], project memory boundaries, JIT credentials.	Prevention of code escape, cross-project data leakage, and credential abuse.

3	Budgets & Economics	LiteLLM proxy integration [5], hierarchical budget tags [9], cost ledger, circuit breakers.	Elimination of bill shock; real-time financial accountability.
4	Scheduling & Resources	Global task queue, token-bucket quotas, pre-emption at checkpoints, priority resolution.	Fair resource distribution; eradication of high-priority task starvation.[2]
5	Governance & Approvals	Policy-as-code (OPA) [7], unified approval inbox, autonomy levels, automated archival.	Safe execution of high-risk workflows with human-in-the-loop oversight.
6	Portfolio Operations	Cross-project dashboards, shared-agent governance, automated incident management and containment.	Ability for one small team to manage dozens of concurrent fleets safely.
7	Intelligent Optimisation	Quality-aware routing, predictive cost forecasting, digital twins, automated rightsizing.	Maximised unit economics and autonomous fleet self-healing.

Features intentionally deferred from MVP (Phases 1-3): Dynamic market-based model auctions, automated cross-project knowledge synthesis, and multi-region active-active execution.

System Invariants

To maintain operational integrity, the platform continuously validates the following non-negotiable invariants:

- Every active fleet belongs to an active, approved project.

- Every running agent possesses a defined, authenticated identity.
- Every model call and execution environment is financially attributable to a specific task and project.[5]
- Every sensitive credential is time-limited and task-scoped.[19]
- Project-private memory is *never* retrievable by another project's agents.[15]
- Mandatory organisation-level policies cannot be weakened by child project configurations.
- Every network egress request defaults to deny.[12]
- Archived projects possess zero active compute and cannot execute tasks.
- Every high-risk (Level 5) action maintains a cryptographic audit trail of human approval.

Final Synthesis

What is the smallest control-plane architecture that allows one person to safely manage multiple projects and agent fleets today, while providing the registries, isolation, budgets, policies, versioning, and economic data needed to evolve later into a full agent operating system?

The optimal minimal architecture is a tripartite system composed of a custom lightweight control-plane database, a dedicated LLM routing proxy, and an external microVM provider.

First, the operator implements a custom control-plane application that hosts the Project, Fleet, and Agent registries. This application acts as the ultimate source of truth, managing task queues, evaluating policy-as-code, and presenting the portfolio dashboard.

Second, all model requests generated by agents are forcibly routed through an intelligent gateway (such as LiteLLM). This off-the-shelf proxy natively handles the complexities of provider neutrality, enforces multi-tenant hierarchical budgets via request tags, trips cost circuit breakers, and captures exhaustive telemetry to populate the economic ledger.[5]

Third, all untrusted agent code execution and external tool utilisation are pushed off the control plane and into ephemeral, hardware-isolated microVMs (such as those provided by E2B or Daytona).[3] These sandboxes are configured with strict default-deny network egress and zero-retention memory rules, guaranteeing cryptographic tenant isolation and GDPR compliance.[12]

This modular, three-pillar foundation provides immediate operational visibility, absolute financial control, and uncompromised security isolation. It requires minimal custom infrastructure engineering, allowing a single non-developer operator to govern dozens of fleets today, while establishing the exact architectural primitives—scheduling, I/O mediation, and resource allocation—required to organically evolve into a fully autonomous, predictive Agent Operating System.

Works cited

1. agiresearch/AIOS: AIOS: AI Agent Operating System - GitHub, accessed July 25, 2026, https://github.com/agiresearch/AIOS
2. AIOS, The LLM Agent Operating System Explained (2026) | Knowlee, accessed

July 25, 2026, https://www.knowlee.ai/blog/aios-llm-agent-operating-system
3. E2B Alternatives: Best Sandbox Environments for 2026 | Blaxel Blog, accessed July 25, 2026, https://blaxel.ai/blog/e2b-alternatives-sandbox-environments
4. How Every Major Tech Company Is Sandboxing AI Agents Differently - Medium, accessed July 25, 2026, https://medium.com/@earlperry562/how-every-major-tech-company-is-sandboxing-ai-agents-differently-f41b65f14d8a
5. LiteLLM cost tracking: Multi-model expense management - Statsig, accessed July 25, 2026, https://www.statsig.com/perspectives/litellm-cost-tracking
6. Multi-Tenant Architecture with LiteLLM, accessed July 25, 2026, https://docs.litellm.ai/docs/proxy/multi_tenant_architecture
7. What Is Open Policy Agent? Best Practices and Use Cases - Orca Security, accessed July 25, 2026, https://orca.security/resources/blog/what-is-open-policy-agent/
8. E2B vs Daytona: Sandbox Comparison for Platform Engineers - ZenML Blog, accessed July 25, 2026, https://www.zenml.io/blog/e2b-vs-daytona
9. Setting Tag Budgets - LiteLLM, accessed July 25, 2026, https://docs.litellm.ai/docs/proxy/tag_budgets
10. AIOS: LLM Agent Operating System - OpenReview, accessed July 25, 2026, https://openreview.net/forum?id=L4HHkCDz2x
11. GitHub - e2b-dev/E2B: Open-source, secure environment with real-world tools for enterprise-grade agents., accessed July 25, 2026, https://github.com/e2b-dev/e2b
12. Internet access - Documentation - E2B, accessed July 25, 2026, https://e2b.dev/docs/network/internet-access
13. E2B Sandbox vs Railway Sandboxes | Blog - Northflank, accessed July 25, 2026, https://northflank.com/blog/e2b-vs-railway-sandboxes
14. Budget Routing - LiteLLM, accessed July 25, 2026, https://docs.litellm.ai/docs/proxy/provider_budget_routing
15. Data Isolation in Agentic AI for Enterprises - Parloa, accessed July 25, 2026, https://www.parloa.com/knowledge-hub/data-isolation-agentic-ai/
16. Upstash Box vs E2B: Two Takes on Computers for AI Agents, accessed July 25, 2026, https://upstash.com/blog/upstash-box-vs-e2b
17. Data Residency Requirements: EU vs US Explained | Secure Privacy Blog, accessed July 25, 2026, https://secureprivacy.ai/blog/data-residency-requirements-eu-vs-us-explained
18. agent-sandbox/agent-sandbox - GitHub, accessed July 25, 2026, https://github.com/agent-sandbox/agent-sandbox
19. Secure code execution - Hugging Face, accessed July 25, 2026, https://huggingface.co/docs/smolagents/en/tutorials/secure_code_execution
20. Multi-tenancy in RAG applications in a single Amazon Bedrock knowledge base with metadata filtering | Artificial Intelligence - AWS, accessed July 25, 2026, https://aws.amazon.com/blogs/machine-learning/multi-tenancy-in-rag-applications-in-a-single-amazon-bedrock-knowledge-base-with-metadata-filtering/
21. EU Data Residency and GDPR Compliance for MCP Servers (2026 Guide) | Truto

Blog, accessed July 25, 2026, https://truto.one/blog/how-to-handle-eu-data-residency-and-gdpr-compliance-for-mcp-servers/
22. AI Agent GDPR Compliance 2026: 12-Point Checklist - Technova Partners, accessed July 25, 2026, https://technovapartners.com/en/insights/security-gdpr-enterprise-ai-agents
23. E2B sandboxes - Docker Docs, accessed July 25, 2026, https://docs.docker.com/ai/mcp-catalog-and-toolkit/e2b-sandboxes/