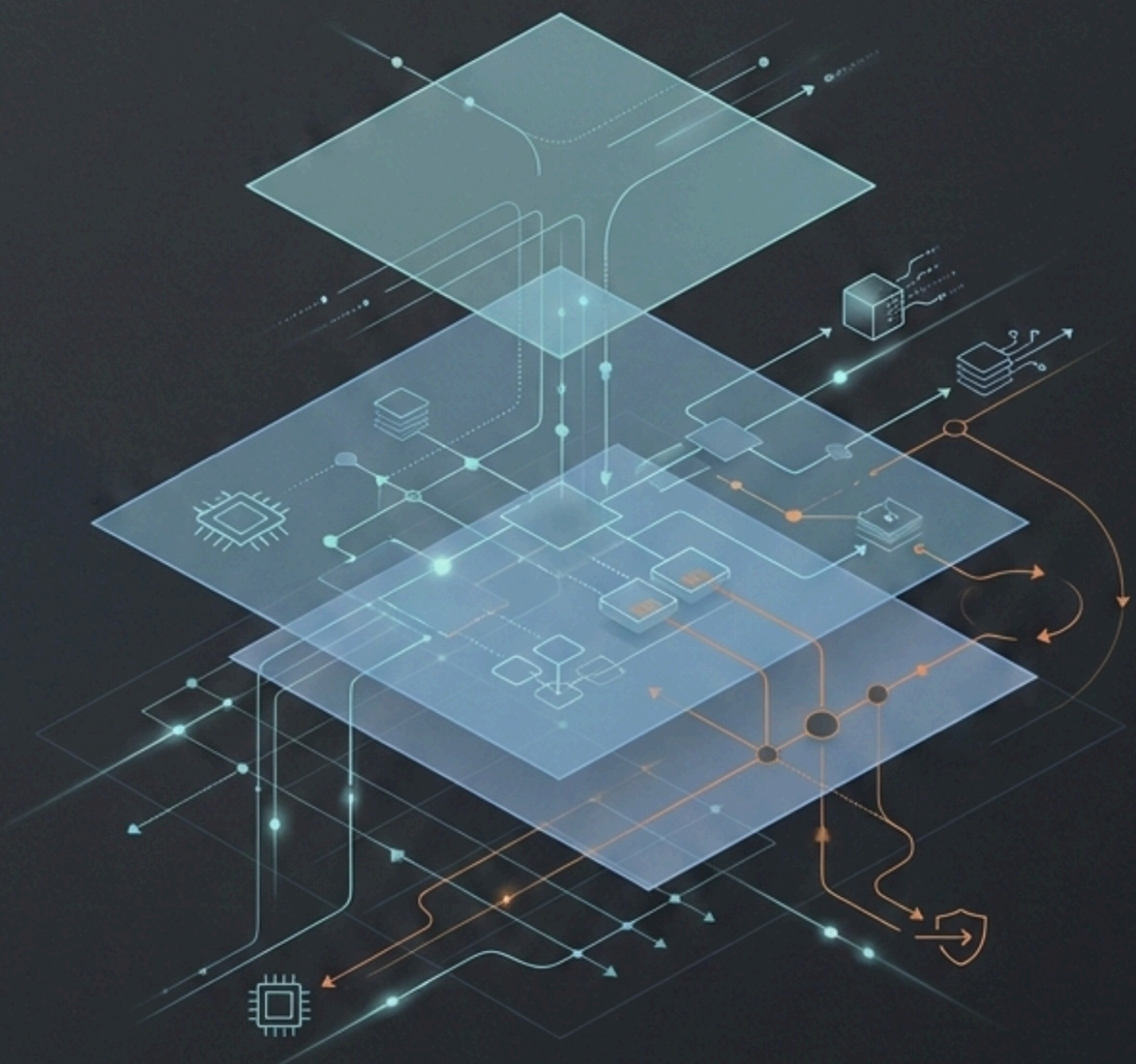
# AI原生基础设施

## 为不确定性而设计的系统



Jimmy Song

# 目录

# 第 1 章

# 什么是 AI 原生基础设施?

> AI 原生基础设施的本质，是让模型行为、算力稀缺与不确定性成为可治理的系统边界。

AI 原生基础设施并非简单的技术清单，而是一套面向“模型成为行为体、算力成为稀缺资产、系统默认不确定”的新秩序。

**AI 原生基础设施的核心不是更快的推理或更便宜的 GPU，而是为模型行为、算力稀缺与不确定性提供可治理、可度量、可进化的系统边界，使 AI 系统在生产环境中可交付、可治理、可演进。**

## 1.1 为什么需要一个更严格的定义

“AI-native infrastructure / architecture”这一术语被越来越多厂商采用，但其含义常常被简化为“更适合 AI 的数据中心”或“更完整的 AI 平台交付”。

在实际应用中，不同厂商对 AI 原生基础设施的理解各有侧重：

- Cisco 强调在 **edge / cloud / data center** 全域交付 AI-native infrastructure，并突出“开放解耦（open & disaggregated）与集成系统（fully integrated systems）并存”的交付路径（如 Cisco Validated Designs）。
- HPE 强调面向 AI 全生命周期、用于模型开发与部署的 **open, full-stack AI-native architecture**。
- NVIDIA 明确提出 **AI-native infrastructure tier**，以支持长上下文与 agentic workload 的 inference context 复用。

对于 CTO/CEO 而言，一个可指导战略与组织设计的定义，需满足以下两点：

- 能阐明 **AI 时代基础设施的第一性约束发生了哪些变化**
- 能将“AI-native”从营销形容词收敛为 **可验证的架构属性与运行机制**

## 1.2 一句话权威定义

AI 原生基础设施（AI-native infrastructure）是：

**以“模型/智能体作为执行主体、算力作为稀缺资产、不确定性作为常态”为前提，通过算力治理把“意图（API/Agent）→ 执行（Runtime）→ 资源消耗（Accelerator/Network/Storage）→ 经济与风险结果”闭环起来的基础设施体系与运行机制。**

这一定义包含两层含义：

- **Infrastructure**：不仅是软硬件堆栈，还包含规模化交付与系统性能力（与厂商普遍强调的“全栈集成/参考架构/生命周期交付”一致）。
- **Operating Model**：它必然改写组织与运行方式，而不仅是技术升级 —— 预算、风险与发布节奏会被强绑定到同一个治理回路中。

## 1.3 三个前提

AI 原生基础设施的核心前提如下，图中给出了三条前提与治理边界的对应关系。

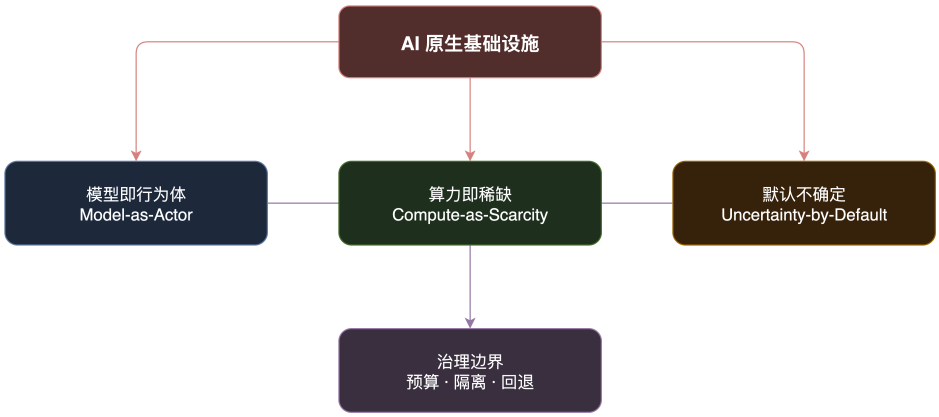



图 1-1: AI 原生基础设施三大前提

- **Model-as-Actor**：模型/智能体成为“执行主体”
- **Compute-as-Scarcity**：算力（加速器、互连、功耗、带宽）成为核心稀缺资产
- **Uncertainty-by-Default**：行为与资源消耗高度不确定（agentic、long-context 场景下更明显）

这三点共同决定：AI 原生基础设施的核心任务并非“让系统更优雅”，而是**在不确定行**

**为下，使系统可控、可持续，并具备规模化交付能力。**

## 1.4 边界：AI 原生基础设施管什么、不管什么

在实际工程中，明确边界有助于聚焦资源与能力建设。下表总结了 AI 原生基础设施的关注点与非关注点：

**不聚焦于：**

- Prompt 设计与业务级 agent 逻辑
- 单个模型的能力与训练秘诀
- 应用层产品功能本身

**专注于：**

- **算力治理（Compute Governance）**：配额、预算、隔离/共享、拓扑与互连、抢占与优先级、吞吐/时延与成本权衡
- **执行形态工程化**：训练/微调/推理/批处理/agentic workflow 的统一运行、调度与观测
- **闭环机制**：意图如何被约束、如何被度量、如何映射到可控的资源消耗与经济/风险结果

## 1.5 可验证的架构属性：三平面 + 一闭环

为便于理解，以下内容介绍 AI 原生基础设施的核心架构属性。

下图给出三平面与闭环的可视化结构，便于在评审时快速对齐边界。

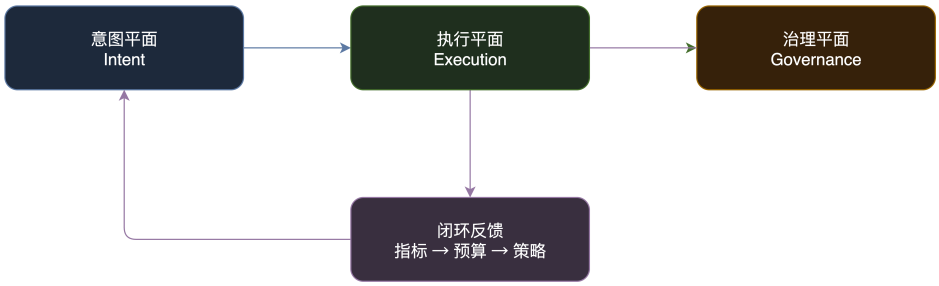



图 1-2: 三平面与一闭环参考架构

**三平面（Three Planes）：**

- **意图平面（Intent Plane）**：API、MCP、Agent workflow、策略表达

- **执行平面（Execution Plane）**：训练/推理/serving/runtime（含工具调用与状态管理）
- **治理平面（Governance Plane）**：accelerator 编排、隔离/共享、配额/预算、SLO 与成本控制、风险策略

**—闭环（The Loop）：**

- **具备“意图 → 消耗 → 成本/风险结果”闭环，方可称为 AI-native。**

这也是 NVIDIA 将 inference context 这类“新状态资产”的共享与复用上升为独立 AI-native 基础设施层的原因：本质上是将 agentic/long-context 造成的资源后果纳入可治理的系统边界。

## 1.6 AI 原生 vs 云原生：差异在哪里

云原生（Cloud Native）关注在分布式环境下，以可移植、可弹性、可观测、可自动化的方式交付服务，其治理对象主要是 **service / instance / request**。

AI 原生基础设施则面向另一组结构性问题：

- **执行单元变化**：从 service 的 request/response，迁移到 agent 的 action/decision/side effect
- **资源约束变化**：从 CPU/内存的可弹性，迁移到 GPU/吞吐/token 的硬约束与成本上限
- **可靠性形态变化**：从“确定性系统的可靠交付”，迁移到“非确定性系统的可控运行”

因此，AI 原生并非“在云原生上加一层模型”，而是将治理中心从 **deployment** 迁移到 **governance**。

## 1.7 落地到工程：AI 原生基础设施需要具备哪些能力

为避免“概念正确、落地失焦”，以下列举了最小闭环能力。

### 1.7.1 资源模型：把 GPU、上下文、token 变成一等资源

云原生将 CPU/内存抽象为可调度资源；AI 原生则需进一步将以下资源纳入治理：

- **GPU/加速器资源**：按切分、共享、隔离、抢占进行调度与治理
- **上下文资源**：上下文窗口、检索链路、缓存命中、KV/推理状态资产复用等，直接影响 token 与成本

- **token/吞吐**：成为可计量的产能与成本载体（可进入预算、SLO 与产品策略）

当 token 成为“产能单位”，平台不再只是运行服务，而是在运营一座“AI 工厂”。

### 1.7.2 预算与策略：把“成本/风险”绑定到组织决策

AI 系统难以采用“上线即完工”的运作方式。预算与策略需成为控制面：

- 预算超标时触发限流/降级
- 风险升高时触发更严格的验证或关闭高风险工具
- 版本发布与实验受“预算/风险余量”约束（将发布节奏制度化）

关键在于**基础设施将组织规则固化为可执行策略**。

### 1.7.3 可观测与审计：把模型行为变成可追责的可观测对象

传统可观测性关注 latency/error/traffic；AI 原生需新增至少三类信号：

- **行为信号**：模型调用了哪些工具、读写了哪些系统、做了哪些动作、造成了哪些副作用
- **成本信号**：token、GPU 时间、缓存命中、队列等待、互连瓶颈
- **质量与安全信号**：输出质量、违规/越权风险、回退次数与原因

缺乏“行为可观测”，治理难以落地。

### 1.7.4 风险治理：把高风险能力纳入持续评估与控制

当模型能力接近“可造成严重伤害”的阈值时，组织需具备成体系的风险治理框架，而非依赖单点提示词或人工 review。

可拆分为两层：

- **系统层可信赖性目标**：对安全、透明、可解释、可追责提出组织级要求
- **前沿能力准备度评估**：对高风险能力进行分级评估、上线门槛与缓释措施

价值在于：将“安全/风险”从理念转化为可执行的发布门槛与运行策略。

## 1.8 Takeaways / Checklist

以下清单可用于判断组织是否已进入 AI 原生阶段：

- 是否将模型视为“会行动的主体”，而非可替换 API?
- 是否将算力与预算纳入业务 SLA 与决策流程?
- 是否将不确定性作为默认前提而非异常?
- 是否存在对模型行为的审计、回滚与责任界定?
- 是否有跨团队的 AI 治理机制，而非单点工程优化?
- 是否能解释系统的运行边界、成本边界与风险边界?

## 1.9 总结

AI 原生基础设施的本质在于：以模型为行为主体、算力为稀缺资产、不确定性为常态，通过治理与闭环机制，实现可交付、可治理、可演进的 AI 系统。只有将这些能力工程化，组织才能真正迈入 AI 原生阶段。

## 1.10 参考文献


- Cisco AI-Native Infrastructure - cisco.com
- HPE AI-native architecture - hpe.com
- NVIDIA Rubin: AI-native infrastructure tier - developer.nvidia.com
- LF Networking: becoming AI-native is a redefinition of the operating model - lfnetworking.org
- NIST AI Risk Management Framework - nist.gov
- Google SRE Workbook - Error Budgets - sre.google
- OpenAI Preparedness Framework - openai.com

# 第 2 章

# AI 原生基础设施一页参考架构：三平面 + 一闭环

> 真正的架构价值，是让复杂系统在 5 分钟内形成组织共识，而不是再造一套新技术栈。

业界主流厂商在表达上各有侧重：Cisco 更强调 AI-native 基础设施与 Cisco Validated Designs 等参考设计与交付体系；HPE 强调面向 AI 全生命周期的 open, full-stack AI-native architecture；NVIDIA 则明确提出为 long-context 与 agentic workloads 的 inference context 复用新增 AI-native infrastructure tier。本章将这些视角收敛为一个**可验证的架构骨架**：三平面 + 一闭环。

本文“架构”是评审骨架，不是组件清单；目的是统一组织语言与评审边界，而不是再造技术栈。

## 2.1 一页架构总览

下方为 AI 原生基础设施三平面与闭环的参考架构详细图，帮助读者快速建立整体认知：

这张图可以理解为：**AI 原生基础设施的新控制面（意图）必须被治理平面约束，并在执行平面产生可计量的资源后果。**

这一点也是厂商叙事差异背后的共同点：Cisco 用参考设计与交付体系把基础设施能力可规模化复制；HPE 用 open/full-stack 覆盖生命周期交付；NVIDIA 则把“上下文状态资产”的复用上升为独立基础设施层。三者都指向同一个问题：**把 AI 的资源后果纳入可治理的系统边界**。

## 2.2 三平面核心能力解析

本节将详细解析三平面各自的核心能力，帮助架构评审时明确关注点。

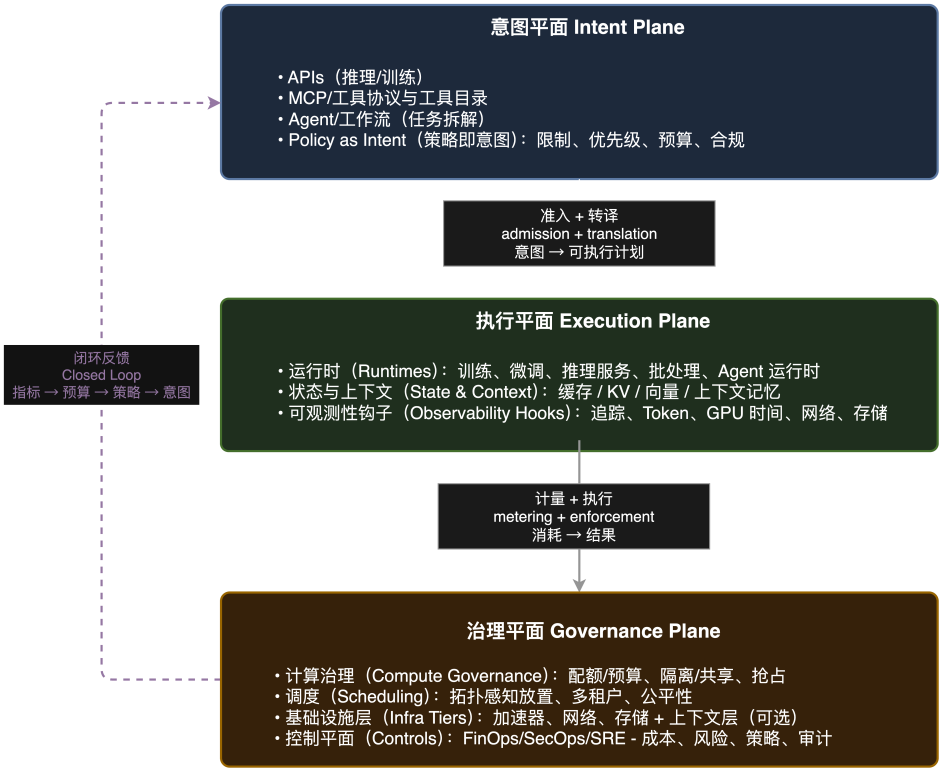



图 2-1: 三平面详细架构图

### 2.2.1 意图平面（Intent Plane）

意图平面负责表达“我想要什么”，包括以下能力：

- 推理/训练 API（入口与契约）
- MCP/工具调用协议与工具目录（把工具访问标准化为“可声明的能力边界”）
- Agent/Workflow（把任务拆解为可执行步骤）
- Policy as Intent：优先级、预算、配额、合规/安全约束（以“意图”的形式前置）

**关键点**：意图平面不是起点本身；真正的起点是 —— **意图能否被转译为可执行且可治理的计划**。否则 Agent/MCP 只会放大不确定性：更多工具、更长链路、更大状态空间、更不可控的资源消耗。

在架构评审时，建议关注以下问题：

- 意图是否可声明（contract）与可拒绝（admission）？
- 意图是否携带预算/优先级/合规约束（policy as intent）？
- 意图到执行的转译是否可追踪（traceable）？

### 2.2.2 执行平面（Execution Plane）

执行平面负责把意图落地为“真实执行”，主要包括：

- 训练、微调、推理 serving、批处理、agentic runtime
- “状态与上下文”服务：缓存/KV/vector/context memory 等，用于承载推理上下文、检索结果、会话状态
- 全链路观测钩子：token 计量、GPU 时间、显存、网络流量、存储 IO、队列等待等

需要强调一个产业趋势：当 long-context 与 agentic workload 普及，“上下文”本身成为关键状态资产，甚至可能上升为独立基础设施层。NVIDIA 在 Rubin 平台中明确提出 inference context memory storage，建立 AI-native infrastructure tier，为 pod 级提供共享的、低延迟的 inference context 以支持复用（面向 long-context 与 agentic workloads）。

评审要点聚焦三件事：

- **执行是否可度量**：是否能在 token/GPU/网络/存储维度做归因?
- **状态是否可治理**：上下文与缓存的生命周期、复用边界、隔离策略是什么?
- **观测是否面向闭环**：观测不是为了“看见”，而是为了“让治理能纠偏”。

### 2.2.3 治理平面（Governance Plane）

治理平面是 AI 原生基础设施的“硬核差异化”，负责把资源稀缺与不确定性变成可控系统：

- 预算/配额/计费：按团队、租户、项目、模型、agent 任务维度治理消耗
- 隔离与共享策略：同卡共享、显存隔离、抢占、优先级、公平性
- 拓扑感知调度：把 GPU、互连、网络、存储拓扑纳入 placement（尤其在训练与高吞吐推理中）
- 风险与合规控制：审计、策略执行点、敏感数据与访问控制
- 与 FinOps/SRE/SecOps 的联动：把成本、可靠性与风险纳入同一套运行机制

从厂商叙事看，这一层通常对应“参考架构 + 全栈交付”：
Cisco 强调在 AI 基础设施中通过“fully integrated systems + Cisco Validated Designs”加速与规模化交付；HPE 以“open, full-stack AI-native architecture”强调端到端交付以支撑模型开发与部署。

治理平面评审的底线问题是：**你能否在预算/风险约束下做可解释的资源分配与降级**

**决策**?

## 2.3 一闭环机制详解

本节介绍闭环机制的核心流程，帮助理解 AI-native 与 AI-ready 的本质区别。

闭环是 AI-native 与“AI-ready”最容易混淆、也是最关键的分界线。

闭环的最小实现包含四步：

1. **Admission（准入）**：在入口就把意图与政策绑定（预算、优先级、合规）
2. **Translation（转译）**：把意图转译为可执行计划（选择 runtime、资源规格、拓扑偏好）
3. **Metering（计量）**：对 token/GPU/网络/存储做端到端计量与归因
4. **Enforcement（执行）**：预算触发降级/限流/抢占；风险触发隔离/审计；SLO 触发扩缩/路由

换句话说：**闭环不是“监控面板”，而是“治理驱动的实时纠偏机制”。**
如果缺乏“意图 → 消耗 → 成本/风险结果”的闭环，系统容易在成本、风险、质量等方向失控。

这也是为什么“AI-native”经常伴随 operating model 的变化：当系统的执行速度与资源消耗都被模型/agent 放大，组织必须把治理机制前置并制度化。LF Networking 也明确指出：成为 AI-native 不只是技术迁移，而是 operating model 的重定义。

## 2.4 一页架构的实际用法

在后续章节中，这张“一页架构”可以反复复用为评审模板：

- 讨论 MCP/Agent：把它们定位到**意图平面**，并用闭环约束（admission/translation）避免“意图泛滥”
- 讨论 runtime 与平台：放到**执行平面**，重点看可观测、可归因、可治理的状态资产（context/cache/KV/vector）
- 讨论 GPU、调度、成本：落到**治理平面**，以预算/隔离/拓扑/计量为抓手
- 讨论企业落地：用**闭环**审视是否“真的 AI-native”（是否能把成本/风险结果回写为可执行策略）

如果你只能记住一句话：

**AI-native 的判定不在“用了多少 AI 组件”，而在“是否存在可执行的治理闭环，把意图约束为可控的资源后果与经济/风险结果”。**

## 2.5 总结

一页参考架构为 AI 原生基础设施提供了统一的系统语言和评审骨架。通过意图、执行、治理三平面与闭环机制，组织能够在架构设计、资源治理和风险控制等方面实现高效协同。未来，随着 AI-native 能力的不断完善，治理闭环将成为企业落地 AI 的核心竞争力。

## 2.6 参考文献


- NVIDIA AI Enterprise Reference Architecture - nvidia.com
- Google Cloud AI Infrastructure - cloud.google.com
- AWS Well-Architected Framework - aws.amazon.com

# 第 3 章

# 为什么必须从算力治理出发，而不是 API 设计

> 算力与治理边界，才是 AI 原生基础设施架构的真正底线。

上一章给出了“三平面 + 一闭环”的一页参考架构。本章将进一步聚焦一个 CTO/CEO 级别的核心问题：

> **AI 原生基础设施到底应当如何分层？哪些属于 API/Agent 的“控制面”，哪些属于 runtime 的“执行面”，哪些必须下沉到“治理面（算力与经济约束）”？**

这个问题之所以关键，是因为过去一年大量“转向 AI”的平台公司，常见的误区是：**把 AI 当成 API 形态变化，而不是系统约束变化**。当你的系统从“服务请求”转向“模型行为”（Agent 的多步行动与副作用），真正决定系统边界的，往往不是 API 设计是否优雅，而是：**算力、上下文与经济约束是否被制度化为可执行的治理边界**。

本章的核心观点可以归纳为：

> **AI 原生基础设施必须从“后果（Consequence）”出发设计，而不是从“意图（Intent）”出发堆叠能力；控制面负责表达意图，但治理面负责限定后果。**

## 3.1 分层的目的：将“意图”与“资源后果”用工程结构绑定

在 AI 原生基础设施中，MCP、Agent、Tool Calling 等机制让系统能力增强的同时，也带来了更高的风险。这里的风险并非抽象意义上的“不可控”，而是工程意义上的“后果不可预算”：

- 行为路径爆炸、长上下文与多轮推理带来资源消耗的长尾；

- 同样的“意图”，可能导致数量级不同的 token、GPU 时间，以及网络/存储压力；
- 若缺乏治理闭环，系统会在“功能更强”的同时走向“成本与风险失控”。

因此，分层的根本目的不是抽象美学，而是实现一个硬约束目标：

> **确保每一层都能把上层“意图”翻译成可执行计划，并产生可计量、可归因、可约束的资源后果。**

换句话说，分层不是为了让架构图更清晰，而是为了把“谁在表达意图、谁在执行、谁在承担后果”写进系统结构里。

## 3.2 AI 原生基础设施五层结构与“三平面”映射

为了帮助理解分层逻辑，下图细化了上一章“三平面”架构，提出了更具落地性的“五层结构”：


图 3-1: 意图到后果的分层治理关系

- 上两层 = **意图平面（Intent Plane）**
- 中间两层 = **执行平面（Execution Plane）**
- 最底层 = **治理平面（Governance Plane）**

下方为五层架构的详细展开，展示了每一层的主要职责和典型能力：

需要特别指出，**MCP 属于 Layer 4（意图与编排层），而非 Layer 1**。原因在于：MCP 主要定义“能力如何暴露给模型/Agent 以及如何被调用”，解决的是控制面的一致性与可组合性，但并不直接负责“能力调用的资源后果如何被计量、约束与追责”。

## 3.3 MCP/Agent 是“新控制面”，但必须被治理层约束

MCP/Agent 之所以被称为“新控制面”，是因为它们将系统的“决策”从静态代码迁移到动态过程：

- “工具目录 + schema + 调用”构成可组合的能力面（capability surface）；

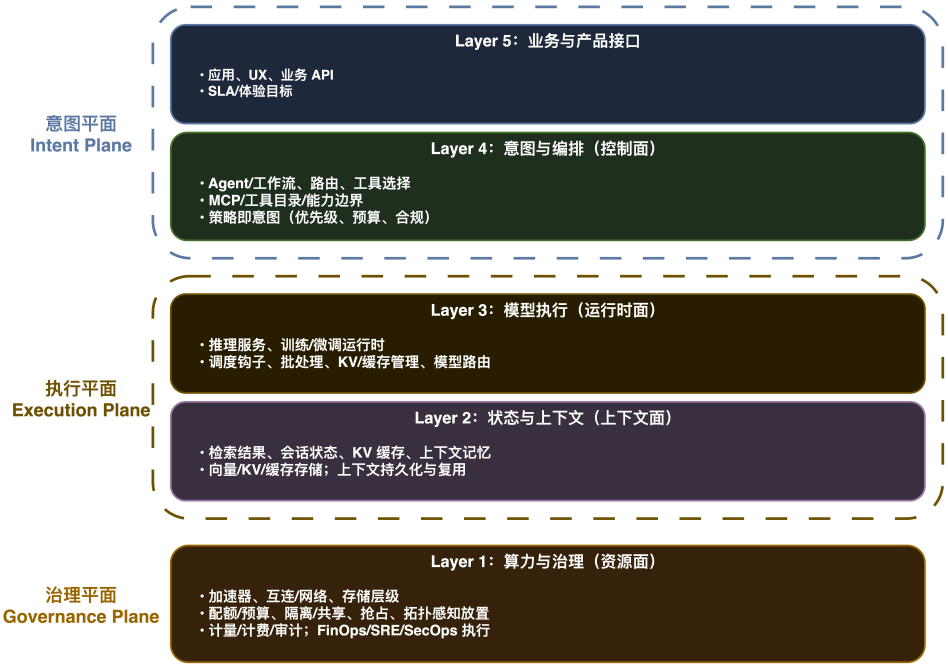



图 3-2: 五层架构图

- Agent 通过选择工具、调用工具、迭代推理来完成任务；
- “策略”不再仅存在于代码分支，而以路由、优先级、预算与合规意图的形式被表达。

但需要强调一条基础设施立场，也是本章的立论基点：

> **MCP/Agent 能表达意图，但 AI 原生的关键在于：意图必须被翻译为可治理的执行计划，并被计量与约束到经济可行的边界内。**

这句话旨在纠正两个常见误区：

- **控制面不是起点**：将 MCP/Agent 视为“AI 平台升级的入口”，容易导致系统走向“能力优先”路径；
- **治理面是底线**：当算力与 token 成为产能单位，任何不受约束的“意图表达”都会以成本、延迟或风险的形式泄漏。

因此，系统分层应明确：Layer 4 负责“表达”，Layer 1/2/3 负责“兑现并承担后果”，而治理闭环负责“纠偏”。

## 3.4 “上下文”正在上升为新的基础设施层

在传统云原生系统中，请求状态大多短命，更多依赖应用层状态管理，基础设施通常只负责“运算与网络”，无需理解“请求上下文”的经济价值。

AI 原生基础设施则不同。long-context、多轮对话、multi-agent 推理让 **推理状态（inference state）经常跨请求存活**，并直接决定吞吐与成本。尤其是 KV cache 与上下文复用，正从“性能优化技巧”演变为“平台产能结构”。

可以总结为一条基础设施规律：

> **当某种状态资产（context/state）成为系统成本与吞吐的决定变量，它就会从应用细节上升为基础设施层级。**

这一趋势已在行业中逐步显现：推理上下文与 KV 复用被明确上升为“基础设施层”能力建设方向。未来还将扩展到分布式 KV、参数缓存、推理路由状态、Agent 记忆等一系列“状态资产”。

## 3.5 AI 原生基础设施的底座：参考设计与交付体系

AI 原生基础设施远不止“买几台 GPU”那么简单。与传统互联网服务相比，AI 工作负载有三大特征，使得“底座”必须更工程化、更产品化：

- **拓扑依赖更强**：网络 fabric、互联、存储层级与 GPU 亲和性决定可用吞吐；
- **稀缺资源更硬**：GPU 与 token 吞吐的边界比 CPU/内存更“不可弹性化”；
- **交付复杂度更高**：多集群、多租户、多模型/多框架并存，只有“可复制交付”才可能规模化。

因此，AI Infra 不只是组件列表，更必须包含“可规模化交付与可重复运行”的体系能力：

**参考设计（validated designs）**

- 将“正确的拓扑与配比”固化为可复用方案。

**自动化交付**

- 将部署、升级、扩缩、回滚与容量规划制度化。

**治理落地**

- 将预算、隔离、计量与审计作为默认能力，而非事后补丁。

从 CTO/CEO 视角，这意味着：你买到的不是“硬件”，而是“可预期产能的交付体系”。

## 3.6 CTO/CEO 视角下的“分层责任边界”

为便于企业内部对齐“谁负责什么、失败的代价是什么”，下表将“技术分层”映射到“组织责任”，避免平台团队只做控制面、却无人承担后果边界。

| 层 | 典型能力 | 主要 Owner（建议） | 失败的代价 |
| --- | --- | --- | --- |
| Layer 5 业务接口 | SLA、产品体验、商业目标 | Product / Business | 客户体验与收入受损 |
| Layer 4 意图/编排（MCP/Agent） | 能力目录、workflow、策略表达 | App / Platform / AI Eng | 行为失控、工具滥用 |
| Layer 3 执行（Runtime） | serving、batching、路由、缓存策略 | AI Platform / Infra | 吞吐不足、延迟抖动 |
| Layer 2 上下文/状态 | KV/cache/context tier | Infra + AI Platform | token 成本飙升、吞吐坍塌 |
| Layer 1 算力/治理 | 配额、隔离、拓扑调度、计量 | Infra / FinOps / SRE | 预算爆炸、资源争用、事故外溢 |

表 3-1: AI 原生基础设施分层与组织责任□□ 射

可以看到，**AI-native 的组织难点不在“有没有 agent”，而在“层间闭环是否建立”**。当模型驱动放大后果，组织必须将治理机制制度化为平台能力：预算可执行、后果可解释、异常可追责、策略可回写。这才是“从算力治理出发”而非“从 API 设计出发”的真实含义。

## 3.7 总结

AI 原生基础设施的分层设计，核心在于将“意图”与“资源后果”工程化绑定。控制面负责表达意图，治理面负责限定后果。只有将治理机制制度化为平台能力，才能在能力提升的同时，确保成本、风险与产能可控。未来，随着上下文、状态资产等新变量的基础设施化，AI Infra 的交付体系也将持续演进，成为企业可持续创新的底座。

## 3.8 参考文献


- Google SRE - Capacity Planning - sre.google
- AWS Well-Architected - Cost Optimization - aws.amazon.com
- Microsoft FinOps for AI - learn.microsoft.com

# 第 4 章

# AI 原生基础设施的运行与治理：指标、预算、隔离、共享、SLO → Cost

> AI 原生基础设施的治理，关键在于如何用制度化方式闭环不确定性带来的成本与风险。

云原生时代，系统运行通常被认为“基本确定”：请求路径可预估、资源曲线大体稳定、扩缩容能及时响应负载变化。然而，进入 AI 时代，这一假设已不再成立 —— **不确定性成为常态**。

本章旨在为 CTO/CEO 提供架构评审的关键结论：

> **AI 原生基础设施的起点，是将不确定性视为默认输入；其目标，是对不确定性带来的资源后果（成本、风险、体验）进行闭环治理。**

这也是“becoming AI-native”在组织语境中逐渐指向运行方式与治理模式的重塑：当系统后果被放大，治理必须制度化。

## 4.1 什么是“不确定性系统”

在本手册中，“不确定性”并非概率论意义上的随机性，而是工程实践中的三类现象：

- **行为不可预测**：执行路径会随模型推理动态变化，尤其在 agentic 过程（Agent 智能体流程）中尤为明显。
- **资源消耗不可预测**：token、KV cache、工具调用、I/O 与网络开销呈现长尾与爆发特征。
- **后果不可线性推导**：同一“意图”可能产生数量级差异的成本与风险结果。

因此，AI 原生基础设施的基础设施问题已从“如何让系统更优雅”转变为：

> **如何在最坏情况发生时，系统仍具备经济可行性、可控性与可恢复性。**

架构评审时，若无法回答“最坏情况是什么、上限在哪里、触发后如何降级/回退”，评审的仍是确定性系统的惯性延伸，而非真正的 AI-native。

## 4.2 不确定性的主要来源

下表总结了 AI 原生基础设施中常见的不确定性来源及其具体表现，便于 CTO/CEO 快速引用。

| 类型 | 具体表现 | 影响层面 |
| --- | --- | --- |
| 行为不确定性 | Agent 任务分解路径变化、工具选择与调用序列变化、失败重试与反思 | 成本、风险、弹性 |
| 需求不确定性 | 并发与 burst、长尾请求、多租户干扰（noisy neighbor） | 资源池、体验、隔离 |
| 状态不确定性 | 上下文跨请求复用、KV cache 迁移与共享 | 性能、成本、治理 |
| 基础设施不确定性 | 网络/存储/互连敏感度高，拥塞与抖动放大为尾延迟 | 体验、成本、稳定性 |

表 4-1: AI 原生基础设施不确定性来源与表现

### 4.2.1 行为不确定性（Behavior Uncertainty）

行为不确定性主要体现在 agent 智能体任务分解路径的变化、工具选择与调用序列的动态调整，以及失败重试、反思（reflection）、多轮规划导致的路径爆炸。工具与上下文通过标准接口组合（如 MCP 协议化接入），系统能力面显著扩大，同时分支空间也成为基础设施治理难题。

更关键的是，工具调用并非“免费的外部函数”，它会占用上下文窗口并消耗 token 预算，放大成本与尾延迟压力。因此，行为不确定性不仅是产品层的“功能弹性”，更是平台层的“成本与风险弹性”，必须被预算化、上限化、可审计化。

### 4.2.2 需求不确定性（Demand Uncertainty）

需求不确定性包括并发与 burst（峰值）、长尾请求（超长上下文、复杂推理）、多租户下的相互干扰（noisy neighbor）。这推动 capacity planning 从“均值容量”转向“尾部容量 + 治理策略”。

在 AI 原生基础设施中，决定体验与成本的往往不是平均请求，而是尾部请求的组合：少量长链路、长上下文、工具密集型请求，足以拖垮共享资源池。因此，需求不确定性要求回答：**哪些请求值得保障、哪些必须限额、哪些应被隔离。**

### 4.2.3 状态不确定性（State/Context Uncertainty）

状态不确定性是 AI 时代最易被低估的一类：**上下文是状态资产**，且常跨请求存在。当推理（inference）的 state / KV cache 被提升为可复用、可共享、可迁移的系统能力时，它不再是应用细节，而是吞吐与单位成本的决定变量。NVIDIA 在公开材料中将 Inference Context Memory Storage 作为新的基础设施层，指向 long-context 与 agentic workload 的状态复用与共享诉求。

结论是：**“上下文/状态”从可选优化，变为基础设施的关键资产，必须可计量、可分配、可治理。**

### 4.2.4 基础设施不确定性（Infrastructure Uncertainty）

AI 负载对网络、互连、存储的敏感度远高于传统微服务负载。拥塞、丢包、I/O 抖动会被放大为 tail latency 与作业完成时间的不稳定，进而在体验与成本上形成“非线性后果”。

这类不确定性通常无法通过“组件选型”解决，而是需要**端到端路径的工程约束**：从拓扑、带宽与队列，到传输协议、隔离策略与拥塞控制，都需纳入治理面，而不仅是运维面。

## 4.3 不确定性如何跨层放大

下图展示了指标、预算与隔离策略构成的闭环关系，用于强调治理必须可回写。

下方流程图展示了不确定性在 AI 原生基础设施中的跨层放大路径：

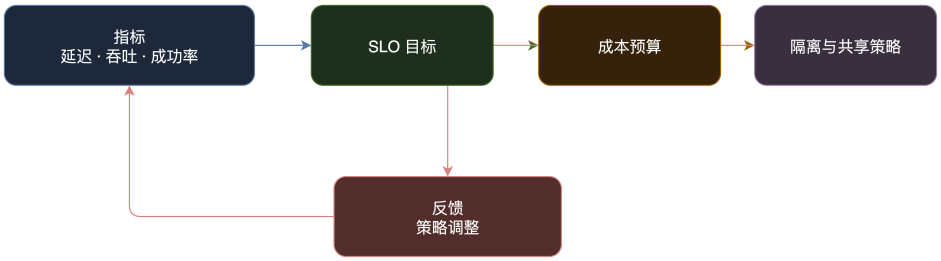



图 4-1: SLO 到成本的闭环关系

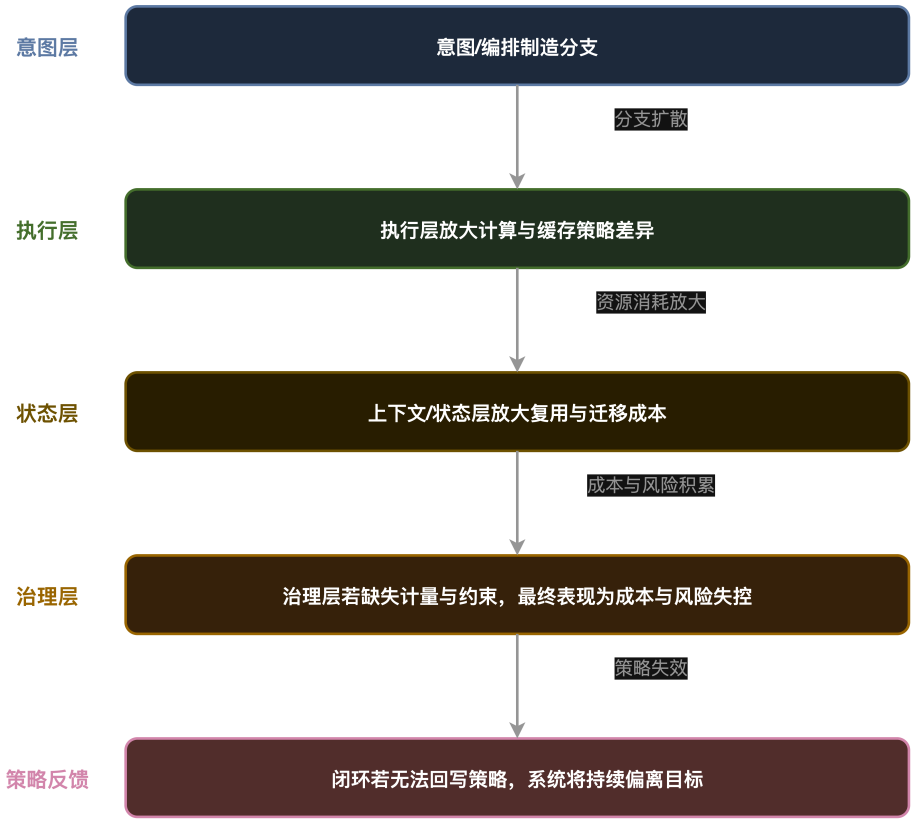



图 4-2: 不确定性跨层放大链路

典型现象包括：

- **Agent 分支爆炸**：工具越多、可组合路径越多，尾部成本越不可控。
- **上下文膨胀**：长上下文与多轮推理让 KV cache 成为性能瓶颈与成本黑洞。
- **资源竞争失真**：多租户下 GPU/网络争用使“平均性能”失去意义，必须治理尾部。

因此，AI-native 的核心不是“让执行更强”，而是让你能稳定回答三个问题：

- **上限在哪里**（预算、步数、调用次数、状态占用）
- **越界怎么办**（降级、回退、隔离、阻断）
- **结果如何回写**（策略迭代与成本纠偏）

## 4.4 AI 原生基础设施的工程响应

企业评审时可参考以下五项“硬标准”，缺一项则无法闭环治理不确定性。

**Admission：入口准入控制**

- 对超长上下文、超大 tool graph、超高预算的请求做分级准入
- 将“预算、优先级、合规”绑定为意图的一部分（policy as intent）
- 明确拒绝理由，解释为何拒绝请求

入口的职责不是“放行功能”，而是将后果约束写进契约。

**Translation：意图转译为可治理执行计划**

- 为请求选择 runtime、路由/批处理策略、缓存策略
- 对 agent 工作流做“上限化”：最大步数、最大工具调用、最大 token
- 纳入可回退路径：确定性替代、缓存答案、人工/规则兜底

从“prompt 驱动执行”升级为“计划驱动执行”，计划必须能被治理面理解与约束。

**Metering：端到端计量与归因**

- 对每个请求/agent 任务的 tokens、GPU time、KV cache footprint、I/O、网络进行计量
- 按租户、项目、模型、工具归因，形成成本与质量口径
- 单独标记“尾部开销”，让长尾不再隐藏在平均值里

没有账本，就没有预算；没有归因，就没有治理，更谈不上 ROI。

**Enforcement：预算与降级机制**

- 预算触发：限流、降级、抢占、排队（按优先级与租户隔离）
- 风险触发：隔离

## 4.5 总结

AI 原生基础设施的治理核心在于将不确定性前置、分层计量、策略反馈与制度化约束，形成成本与风险的闭环。只有具备 Admission、Translation、Metering、Enforcement 等工程机制，才能在不确定性常态下实现经济可行、可控、可恢复的系统运行。

## 4.6 参考文献


- Google SRE Book - Service Level Objectives - google
- FinOps Foundation - AI Cost Management - finops.org
- OpenAI Usage Policies - openai.com

# 第 5 章

# 组织与文化：Operating Model 如何变化

> 算力治理闭环，是 AI-native 组织可持续创新的底层保障。

API / Agent / MCP 解决的是“意图如何表达”；算力治理解决的是“意图的资源后果是否经济可行、风险可控”。在 AI 时代，后者成为前者的前置条件。治理缺失的 API-first，只会放大成本与不确定性，把组织推向“功能可用但不可持续”的陷阱。

FinOps Foundation 在《Scaling Kubernetes for AI/ML Workloads with FinOps》中直言：Kubernetes 的弹性极易演化为 **runaway cost problem**。因此，FinOps 不应只是成本报表，而必须成为共同的 operating model，让每一次扩展决策都同时回答两件事：**性能 SLO 是否满足**，以及 **是否负担得起**。

## 5.1 API-first 的“隐含假设”在 AI 时代的挑战

下图展示平台、ML 与安全团队的责任边界与责任链关系。

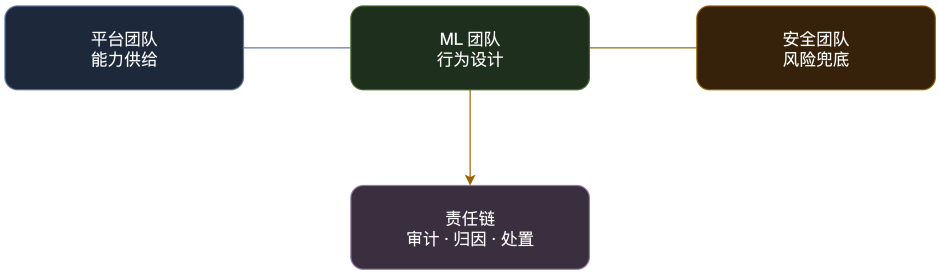



图 5-1: 组织责任边界与责任链

API-first 的直觉路径是：先跑通接口与工作流，随后再通过工程化逐步优化性能与成本。在 AI 原生基础设施中，这条路径经常失效，因为它依赖三条在 AI 时代已不再成立的隐含假设。

**假设一：资源不是核心稀缺**

传统软件将稀缺性押注在工程效率、吞吐、稳定性；而 AI 原生基础设施的稀缺性首先来自 **GPU / 互连 / 功耗** 这些资产边界。稀缺不再是“扩容慢”，而是“扩不动且贵”，并且受供应链与机房条件共同约束。

**假设二：请求成本可预测**

传统请求的成本分布相对稳定；AI 请求则天然长尾：agentic 任务的分支、长上下文的膨胀、工具调用的链式放大，都会让 token 与 GPU time 变成不可线性外推的随机变量。你以为在扩“QPS”，实际上在扩“尾部概率事件的总成本”。

**假设三：状态是短命、可丢弃的**

云原生时代强调无状态扩展，将状态外置；但在推理侧，**推理状态/上下文复用** 常常决定单位成本是否可控。NVIDIA 在 Rubin 的 ICMS（Inference Context Memory Storage）中将其描述为“新推理范式带来的 context storage challenge”：KV cache 需要跨会话/跨服务复用，sequence length 增长导致 KV cache 线性膨胀，迫使其持久化与共享访问，形成“新的 context tier”，并用 TPS 与能效收益证明这不是锦上添花，而是规模化的门槛。

在 AI 原生基础设施中，状态与算力治理已成为“能否运行”的前置条件，而非后续优化项。

## 5.2 算力治理的本质：治理的对象是什么

“算力治理”常被误解为“管理 GPU”，但真正要治理的是 **意图的资源后果**。更准确地说，治理的是四类对象的组合效应：

**Token 经济（Token Economics）**

- 每个请求/任务的 token 消耗、上下文膨胀、工具定义与中间结果带来的隐性 token 税，最终直接映射为成本与延迟。

**加速器时间（Accelerator Time）**

- GPU time、显存占用、批处理策略、路由与缓存命中对有效吞吐的影响。关键不在于“有没有 GPU”，而是“单位 GPU 时间产出是否可控”。

**互连与存储（Fabric & Storage）**

- 训练 all-reduce、推理 KV/cache 共享、跨服务数据移动带来的网络与存储压力。AI 的性能与成本往往被 fabric 放大，而不是被 API 放大。

**组织预算与风险（Budget & Risk）**

- 多租户隔离、公平性、审计、合规与可追责。这些决定系统能否扩展到多个团队/业务线，而不仅是把 demo 扩到更多实例。

FinOps Foundation 也强调：AI/ML 的成本驱动不仅是 GPU，存储（checkpoints/embeddings/artefacts）、网络（分布式训练/跨 AZ）以及额外许可与 marketplace 费用，常常会“悄悄超过算力”。因此，治理对象必须覆盖端到端，而不是只盯推理账单。

## 5.3 MCP/Agent：治理缺失下的放大效应

MCP/Agent 扩大的是“能力面”，但同时会让成本曲线更陡，尤其在治理缺失时表现为指数级放大：

- **工具越多，分支越多**：计划空间变大，长尾概率上升，成本波动不可控。
- **工具定义与中间结果占用上下文**：直接消耗 context window 与 token，转化为成本与延迟。
- **更强的工具使用能力触发更多外部 I/O**：外部系统调用、网络往返、数据搬运都会进入整体成本函数。

Anthropic 在“Code execution with MCP”中明确指出：直接工具调用会因工具定义与中间结果占用 context window 而增加成本与延迟；当工具数量上升到几百上千时，会成为规模化瓶颈，因此提出用代码执行形态提升效率、减少 token 消耗。

在 MCP/Agent 时代，治理不是压制创新，而是让创新在预算边界内可持续。治理缺失时，Agent 不是生产力工具，而是成本放大器。

## 5.4 “算力治理优先”的最小实现路径

你可以不绑定任何厂商，但必须实现一个“最小可行治理堆栈”。目标不是完美，而是让系统从第一天就具备可控的边界条件。

**准入与预算（Admission + Budget）**

- 为工作负载类型（training / inference / agent tasks）设定预算与优先级。
- 将预算、最大步数、最大 token、最大工具调用次数纳入 policy-as-intent，并在入口强制执行。

FinOps 的核心观点是：将 FinOps 早期嵌入架构，让每次扩展同时回答“性能”与“可负担性”，否则账单只会在事故发生时才被关注。

**端到端计量与归因（Metering + Attribution）**

- 至少做到一条可追溯链路：request/agent → tokens → GPU time/显存 → 网络/存储 → 成本归因（租户/项目/模型/工具）。
- 没有归因，就没有治理；没有治理，企业内就无法规模化，因为成本与责任无法对齐，组织会在“谁消耗了预算”上内耗。

**隔离与共享（Isolation + Sharing）**

- **共享** 用于提高利用率；**隔离** 用于降低风险。两者必须同时存在，而不是二选一。
- CNCF 的 Cloud Native AI 报告指出：GPU 虚拟化与共享（如 MIG、MPS、DRA 等）能提升利用率、降低成本，但需要谨慎编排与管理，并要求 AI 与云原生工程团队协作。
- 治理的关键不是选择共享还是隔离，而是将其变成可执行策略：谁在什么条件下共享，谁在什么条件下隔离。

**拓扑与网络作为一等公民（Topology + Fabric First）**

- AI 训练与高吞吐推理对网络特性高度敏感。
- Cisco 的 AI-ready 基础设施设计指南与相关 CVD/Design Zone 强调：为 AI/ML 工作负载构建高性能、lossless Ethernet fabric，并通过 validated designs 交付参考架构与部署指南。
- 这意味着拓扑不是“机房同学的事”，而是决定 JCT、尾延迟与容量模型是否成立的核心变量。

**上下文/状态成为治理对象（Context as a Governed Asset）**

- 当 long-context 与 agentic 成为主流，KV cache 与 inference context 的复用将直接决定单位成本。
- NVIDIA 的 ICMS 将其定义为“新的 context tier”，用于解决 KV cache 复用与共享访问，并强调 TPS/能效收益。
- 在这个时代，把上下文当作临时变量，是主动放弃成本控制权。

## 5.5 反模式清单

以下反模式并非“工程不优雅”，而是在组织层面会触发失控，值得警惕。

**API-first，把治理当后续优化**

- 结果：系统先上线，后发现单位成本与尾延迟不可控，只能通过限功能/限流进行“硬刹车”，最终把产品路线锁死。
- 对照：FinOps 指出弹性很容易变成 runaway costs，必须将成本治理提前进入架构决策。

**把 MCP/Agent 当能力加速器，不把它当成本放大器**

- 结果：工具越多越“聪明”，但 token 与外部调用成本指数上升，工程团队被迫用“更复杂的提示词与规则”去对抗系统性放大。
- 对照：Anthropic 指出工具定义与中间结果会占用上下文、增加成本与延迟，并提出更高效的执行形态作为规模化路径。

**只买 GPU，不做共享/隔离与编排**

- 结果：利用率低、争用严重、预算爆炸，组织内部互相指责“谁在抢资源、谁在烧钱”。
- 对照：CNCF Cloud Native AI 报告强调共享/虚拟化能提升利用率，但必须配套编排与协作机制。

**忽视网络与拓扑，把 AI 当普通微服务**

- 结果：训练 JCT 与推理 tail latency 被网络放大，容量规划与成本模型失效，扩容越多越不稳定。
- 对照：Cisco 在 AI-ready 网络设计与 validated designs 中将 lossless Ethernet fabric 等要求作为 AI/ML 的关键基础。

## 5.6 总结

AI-native 的第一性入口是算力治理闭环：预算与准入、计量与归因、共享与隔离、拓扑与网络、上下文资产化。API / Agent / MCP 依然重要，但必须被这套闭环约束，否则系统只能在“更聪明”与“更破产”之间摇摆。

## 5.7 参考文献


- MIT Sloan - sloanreview.mit.edu
- Google Cloud - cloud.google.com
- OECD AI Principles - oecd.ai

# 第 6 章

# 迁移路线图：从云原生到 AI 原生

> 迁移不是“重建平台”，而是用治理闭环和组织合同，把不确定性转化为可控的工程能力。

前五章已经明确：**AI 原生基础设施是 Uncertainty-by-Default（不确定性为常态）**。因此，架构起点必须是 **算力治理闭环**，而非“接入模型即完成迁移”。否则，系统极易在 **成本**（runaway cost）、**风险**（越权/副作用）、**尾部性能**（P95/P99 与队列尾巴）三个维度失控。

这也解释了 FinOps Foundation 强调：在 Kubernetes 上跑 AI/ML，“弹性”极易演化为不可控的成本外溢。**FinOps 必须作为共享 operating model 前置进入架构与组织，而非事后对账。**

本文将给出一份**可执行的迁移路线图**，涵盖技术演进路径与组织落地方式。你无需一次性“重建 AI 平台”，但必须在每个阶段跑通**治理闭环**：预算/准入、计量/归因、共享/隔离、拓扑/网络、上下文资产化。

## 6.1 迁移的北极星：从交付平台到治理闭环

下图展示从旁路试点到 AI-first 重构的迁移路径。


图 6-1: AI 原生迁移路线图

云原生迁移通常以“交付能力”为中心：CI/CD、自助平台、服务治理、弹性伸缩。其默认假设是系统确定、成本随请求线性增长、扩缩容不会显著改变系统边界。

**AI 原生迁移则必须以“治理闭环”为中心**，聚焦成本、风险、尾部性能、状态资产。其默认假设恰好相反：系统天然不确定，推理/Agent 的“动作与后果”会将成本与风险带

入非线性区域。

**AI-Native Migration = 建立 AI Landing Zone + Compute Governance Loop + Context Tier，并让所有 agent/API/runtimes 在该闭环内运行。**

这里将“Landing Zone”提升到北极星层级，不是追热点，而是因为它天然承担一项组织级任务：**划清平台团队与工作负载团队的责任边界**。主流云厂普遍用 Landing Zone 承载“共享治理基线”（网络、身份、策略、审计、配额/预算），业务团队则在受控边界内自助迭代应用。对 AI 而言，这种边界是治理闭环的承载体。

## 6.2 迁移前置条件：先补三块地基，再谈应用爆发

你可以并行做 PoC、做应用，但如果这三块地基缺失，任何“应用爆发”都可能转化为平台救火与财务争议。

**地基 A：FinOps / Quotas as Control Plane（财务与配额是控制面）**

迁移第一步不是“上线第一个 Agent”，而是将**预算、告警、showback/chargeback、配额**纳入基础设施控制面：

- 预算与告警不仅是财务报表，更是运行时策略的触发器（限流、降级、排队、抢占）。
- showback/chargeback 不只是算账，更是将“成本后果”绑定到组织决策与产品边界。
- 配额不是静态限制，而是可演进的治理手段（按租户/团队/用例的动态预算与优先级）。

如果无法将每个 agent/job 的主要消耗归因到 team/project/model/use-case（至少覆盖 tokens、GPU time、KV footprint、关键网络/存储），则尚未达到“规模化”起跑线。可以试点，但不宜扩展。

**地基 B：Resource Governance（GPU 共享/隔离与编排能力）**

AI 原生基础设施的“弹性”受制于稀缺算力的治理方式。将 GPU 当普通资源，结果往往是**利用率低**与**争用失控**。因此需具备可落地的共享/隔离与编排能力组合：

- **共享/切分**：MIG/MPS/vGPU 等路线，让“独占”变为“池化”。
- **调度升级**：引入拓扑、队列、公平性、抢占、成本等级的显式建模。
- **编排闭环**：将隔离、抢占、优先级策略固化为可执行规则。

关键不在于选哪种切分技术，而在于能否将 GPU 从“机器资产”提升为**一等治理资源**，并纳入预算与准入体系。

**地基 C：Fabric as a First-Class Constraint（网络/互连是一级约束）**

训练与高吞吐推理对拥塞、丢包、尾延迟极度敏感。忽视网络与拓扑，易导致“看似偶发、实则结构性”的问题：

- 训练 JCT 被尾部放大，容量规划失效；
- 推理 P99 与排队尾巴被放大，SLO 难以兑现。

因此需构建可复用的 AI-ready 网络基线：容量假设、lossless 策略、隔离域划分、测量与验收口径。网络不是“后面再优化”，而是 Day 31 – 60 必须落地的基线工程。

## 6.3 迁移路径选择：按组织风险与技术债务分层

迁移不是“选一条路走到黑”，而是将不同风险偏好与债务结构的组织，映射到不同起步方式与退出标准。可并行推进，但需为每条路径定义**适用条件**与**退出标准**。

**路径一：旁路试点（Bypass Pilot / Skunkworks）**

适用于云原生平台稳定运行，但 AI 需求刚起、组织不确定性高、治理机制尚未成型的场景。

做法是在现有平台旁建立“AI 最小闭环沙箱”，目标不是“功能齐全”，而是“闭环跑通”：

- 独立 GPU 池（或至少独立队列）+ 基本准入与预算
- 最小 token/GPU metering 与归因
- 受控推理/agent 入口（max context / max steps / max tool calls）
- “失败可接受”的 SLO 与成本上限（先定义边界，再谈体验）

退出标准：

- 成本曲线可解释（至少能归因到团队/用例）
- GPU 利用率与隔离策略形成可复用模板
- 试点能力可下沉为平台能力（进入路径二）

**路径二：分域隔离的平台化（Domain-Isolated Platform）**

适用于 AI 已进入多团队、多租户阶段，需要将“试点资产”沉淀为平台能力，防止成本与风险跨域扩散。

做法是建设 AI Landing Zone，由平台团队集中管理共享治理能力，工作负载团队在受控边界内自助迭代应用。

平台侧必备模块（建议按“治理闭环”组织）：

- **Identity/Policy**：统一身份、策略下发与审计（policy-as-intent）
- **Network/Fabric baseline**：AI-ready 网络基线与自动化验收
- **Compute governance**：配额、预算、抢占、公平性、隔离/共享
- **Observability & Chargeback**：端到端计量、告警、showback/chargeback
- **Runtime catalog**：推理/训练 runtime 的“黄金路径”与模板化交付

退出标准：平台提供“可复制的 AI 工作负载落地方式”，并能在预算约束下扩展用例数量，而非靠人工救火维持稳定。

**路径三：AI-First 重构（AI Factory / Replatform）**

适用于 AI 已是核心业务，需要将基础设施当作“生产线”而非“集群”，并将优化目标从“上线功能”切换为“吞吐/单位成本/能效”。

做法围绕“状态资产 + 单位成本”重构：

- 推理/agent 的 **context/state** 被显式治理与复用（不再是应用内技巧）
- 引入 **Context Tier** 架构假设：长上下文与 agentic 推理要求 inference state / KV cache 能跨节点、跨会话复用
- 用“单位 token 成本、尾部延迟、吞吐/能效”驱动平台演进，而非“新增组件数量”

退出标准：能持续用“单位成本与尾部性能”做工程决策，并将上下文复用当作平台能力，而非应用团队的技巧性缓存。

## 6.4 90 天可执行计划：AI Landing Zone + 最小治理闭环

目标是在 90 天内跑通“AI Landing Zone + 最小治理闭环”，形成可复制模板。关键不是覆盖所有场景，而是打通**准入—计量—执行—反馈**闭环。

**Day 0 – 30：账本立起来（Cost & Usage Ledger）**

首先需定义归因维度，建立 budgets/alerts 与基线报表，并落地 quotas / usage controls。

- 归因维度：tenant/team/project/model/use-case/tool
- 建立预算与告警、基线报表（成本 + 业务价值指标）

- 落地配额与用量控制（至少覆盖 GPU 配额与关键服务配额）

交付物：

- 成本与用量看板（周报级、可追溯）
- “准入策略 v0”（max context / max steps / max budget）

**Day 31 – 60：资源治理立起来（GPU Governance + Scheduling）**

此阶段需评估 GPU 共享/隔离策略，引入拓扑/网络约束，形成推理与训练两类黄金路径。

- GPU 共享/隔离策略：MIG/MPS/vGPU/DRA 路线评估与 PoC（以可执行策略为验收）
- 引入拓扑/网络约束，形成 AI-ready 网络基线与容量假设（含验收口径）
- 形成推理/训练两类模板化交付路径

交付物：

- 工作负载模板（推理与训练各 1 个）
- 调度与隔离策略（白名单化、可审计）

**Day 61 – 90：闭环立起来（Enforcement + Feedback）**

最后阶段需执行预算策略，将试点用例迁移到 landing zone，并固化组织接口。

- 执行预算：限流/排队/抢占/降级策略，并与 SLO 联动
- 迁移试点用例到 landing zone（或将 landing zone 能力服务化）
- 固化“组织接口”：平台团队 vs 工作负载团队责任边界（形成可执行合同）

交付物：

- “AI 平台运行手册 v1”（含 oncall、变更、成本审计）
- 两条可复制的用例落地路径（新增用例 ⩽ 30 分钟上线黄金路径）

## 6.5 Operating Model：平台团队与工作负载团队的“合同”

迁移能否成功，取决于是否建立了清晰、可执行的“组织合同”。合同本质是：谁为“能力供给”负责，谁为“行为后果”负责。

**平台团队提供（必须稳定）**

Landing zone、网络基线、身份与策略、预算/配额体系、计量归因、GPU 治理能力、运行时黄金路径

**工作负载团队负责（必须自助）**

模型选择、prompt/agent 逻辑、工具接入、SLO 定义、业务价值度量、用例风险分级与回退路径

这也是 FinOps Framework 强调 operating model（personas、capabilities、maturity）而不仅是工具的原因：没有“合同”，预算难以执行；预算无法执行，闭环难以成立。

## 6.6 迁移反模式

以下为常见迁移反模式及其后果：

| 反模式 | 典型后果 |
| --- | --- |
| 只建 API/Agent 平台，不建账本与预算 | runaway cost（最常见，且事后补救难度大） |
| 把 GPU 当普通资源，不做共享/隔离与调度升级 | 利用率低 + 争用失控，平台被迫以“行政手段”分配算力 |
| 忽视网络与拓扑 | 尾部延迟与训练 JCT 被放大，容量规划失效，SLO 难以兑现 |
| 上下文不资产化（只在应用里做“技巧性缓存”） | 长上下文/agentic 时代单位成本失控，复用能力难以平台化沉淀 |

表 6-1: 常见迁移反模式与后果

## 6.7 总结

AI 原生迁移的核心，不是“迁移清单”，而是**在不确定性前提下，将成本、风险、尾部性能纳入同一治理闭环，并用 Landing Zone 承载组织合同，用 Context Tier 实现状态复用的基础设施能力**。只有这样，才能让平台与业务在规模化演进中保持可控与高效。

## 6.8 参考文献


- McKinsey on AI Strategy - mckinsey.com
- Thoughtworks Technology Radar - thoughtworks.com
- Google Cloud Adoption Framework - cloud.google.com

# 第 7 章

# 术语表

> 统一术语是组织形成共识的第一步。

结论先行：在 AI 原生基础设施语境中，关键术语必须保持一致，否则治理与沟通都会失焦。

以下术语表用于跨团队对齐。

## 7.1 核心术语

**AI 原生基础设施 / AI Native Infrastructure**

以“模型/智能体作为执行主体、算力作为稀缺资产、不确定性作为常态”为前提，通过算力治理把“意图 → 执行 → 资源消耗 → 经济与风险结果”闭环起来的基础设施体系。

**模型行为体 / Model-as-Actor**

模型/智能体成为“执行主体”，具备行动能力，会调用工具、修改系统状态、产生副作用，因此需要治理与审计。

**算力稀缺 / Compute-as-Scarcity**

算力（GPU、互连、功耗、带宽）成为核心稀缺资产，扩容受供应链与机房条件约束，成本不可弹性化。

**默认不确定 / Uncertainty-by-Default**

行为与资源消耗高度不确定（尤其在 agentic、long-context 场景下），需要验证与回退机制。

**意图平面 / Intent Plane**

API、Agent、策略表达层，负责表达“我想要什么”，包括优先级、预算、合规等策略。

**执行平面 / Execution Plane**

训练/推理/serving/runtime 层，负责把意图落地为真实执行，包括状态管理、工具调用、模型路由等。

**治理平面 / Governance Plane**

配额/预算、隔离/共享、成本控制层，负责限定资源后果，包括拓扑感知调度、SLO 与风险策略。

**闭环 / The Loop**

具备“意图 → 消耗 → 成本/风险结果”闭环，包括四个步骤：Admission（准入）、Translation（转译）、Metering（计量）、Enforcement（执行）。

**算力治理 / Compute Governance**

治理意图的资源后果，包括四类对象：Token 经济、加速器时间、互连与存储、组织预算与风险。

**FinOps / Financial Operations**

将成本治理早期嵌入架构，让每次扩展决策同时回答“性能是否满足”与“是否负担得起”。

**Agent / 智能体**

通过选择工具、调用工具、迭代推理来完成任务的行为主体，其行为路径与资源消耗具有不确定性。

**MCP / Model Context Protocol**

把工具访问标准化为“可声明的能力边界”的协议，定义能力如何暴露给模型/Agent 以及如何被调用。

**运行模型 / Operating Model**

组织与运行方式的制度设计，包括责任边界、协作机制与决策流程，回答“谁负责什么、失败的代价是什么”。

## 7.2 参考文献


- ISO/IEC 22989 AI Concepts and Terminology
- NIST AI Glossary
- OECD AI Glossary

# 第 8 章

# 检查清单（10 问）

以下 10 个问题用于判断组织是否具备 AI 原生基础设施的战略与执行准备。下图将问题分为战略、治理、执行三类，便于会议讨论。

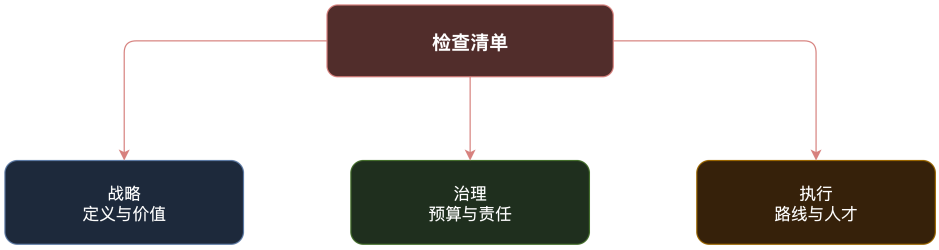



图 8-1: 检查清单结构图

1. 你能否把每个主要 AI 工作负载的 **单位成本** 定义清楚（例如：每 1M tokens、每次 agent 任务、每个 batch 作业）？
2. 你是否具备 **预算/配额机制**，能把团队/项目/租户的算力消耗限制在可控范围？
3. 你是否能在“性能（吞吐/时延）—成本—风险”之间做**显式的政策选择**（而不是靠口头约束）？
4. 你的平台是否能处理 **不确定性**：峰值、长尾、agent 路径爆炸导致的资源波动？
5. Agent/MCP 的“意图”是否被映射到 **可执行且可计费/可审计的资源后果**？
6. 你是否有明确的 **资源隔离与共享策略**（同卡共享、显存隔离、抢占、优先级）来提升利用率？
7. 你能否做到跨层可观测：从 **请求/agent → runtime → GPU/网络/存储 → 成本** 的端到端链路？
8. 你的基础设施是否支持快速引入新硬件/互连/拓扑变化（异构与演进是常态）？
9. 组织层面是否建立了 **“AI SRE / ModelOps + FinOps”** 的协作机制与责任边界（谁为成本与可靠性负责）？

10. 当你说“我们是 AI-native”，你是否能在一页纸上给出 **三平面 + 一闭环** 的架构图与治理策略?

## 8.1 参考文献


- Harvard Business Review - AI Strategy
- MIT Sloan - Executive Guide to AI
- World Economic Forum - AI Governance