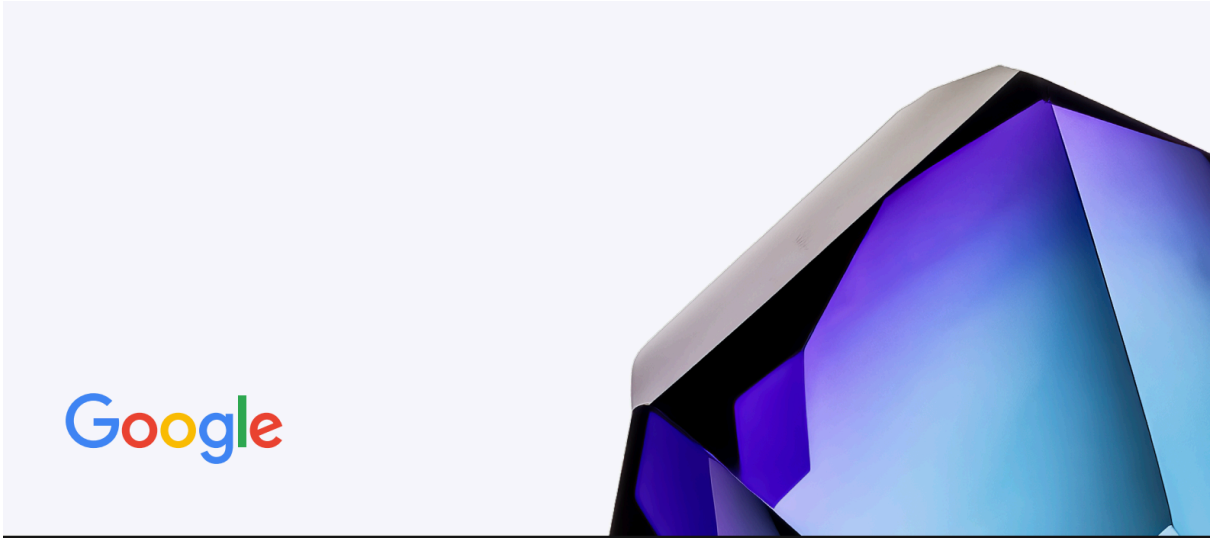
从原型到生产

作者： Sokratis Kartakis, Gabriela Hernandez Larios, Ran Li, Elia Secchi, and Huang Xia

翻译：Google notebooklm

英文版链接：
https://drive.google.com/file/d/1s00Cr_C8LXtrsGrlRG4WUJx4GmAtdzrQ/view



> 构建一个**智能体**很容易。信任它却很难。

摘要

这份白皮书提供了关于 **AI 智能体**运营生命周期的**综合技术指南**，重点关注**部署、扩展和生产化**。本指南以第四天关于**评估和可观测性**的内容为基础，强调如何通过**稳健的 CI/CD 管道**和**可扩展的基础设施**，建立必要的**信任**，将**智能体**投入生产环境。它探讨了将**基于智能体**的系统从**原型过渡到企业级解决方案**所面临的挑战，并特别关注了 **Agent2Agent (A2A) 互操作性**。本指南为 **AI/ML 工程师、DevOps 专业人员和系统架构师**提供了**实用的见解**。

引言：从原型到生产

你可以在几分钟，甚至几秒钟内，启动一个 **AI 智能体**原型。但是，将那个巧妙的演示转化为你的业务可以依赖的、值得信赖的、**生产级系统**呢？那才是真正工作的开始。欢迎来到“最后一公里”的生产差距，我们与客户的实践中持续观察到，大约 **80% 的精力**并非花在**智能体**的核心智能上，而是花在使其**可靠和安全**所需的基础设施、安全和验证上。

跳过这些最后步骤可能会导致几个问题。例如：

- 一个客服**智能体**被人诱骗免费赠送产品，因为你忘记设置正确的防护栏。
- 用户发现他们可以通过你的**智能体**访问机密的内部数据库，因为身份验证配置不当。
- 一个**智能体**在周末产生了巨额的消耗账单，但没人知道原因，因为你没有设置任何监控。
- 一个昨天运行完美的关键**智能体**突然停止工作，但你的团队却手忙脚乱，因为没有持续评估机制。

这些不仅仅是技术问题；它们是重大的业务失败。尽管 DevOps 和 MLOps 的原则提供了重要的基础，但仅凭它们是不够的。部署**智能体**系统引入了一类新的挑战，需要我们对运营规程进行演进。与传统 ML 模型不同，**智能体是自主交互的、有状态的，并遵循动态执行路径**。这带来了独特的运营难题，需要专业的策略：

- **动态工具编排： 智能体**的“轨迹”是在其选择和使用工具时即时组装的。这要求一个每次行为都不同的系统，需要稳健的**版本控制、访问控制和可观测性**。

- **可扩展的状态管理： 智能体**可以记住跨交互的内容。在大规模上安全且一致地管理会话和内存 是一个复杂的系统设计问题。
- **不可预测的成本和延迟： 智能体**可以通过许多不同的路径来寻找答案，这使得其成本和响应时间在没有智能预算和缓存的情况下极难预测和控制。

为了成功应对这些挑战，你需要一个建立在**三个关键支柱**之上的基础：**自动化评估**、**自动化部署 (CI/CD)** 和**全面的可观测性**。

这份白皮书是你构建该基础并迈向生产环境的分步手册！我们将从投产前的要素开始，展示如何设置**自动化 CI/CD 管道**，并利用**严格的评估**作为关键的质量检查。然后，我们将深入探讨在实际环境中运行**智能体**的挑战，涵盖扩展、性能调优和**实时监控**的策略。最后，我们将展望令人兴奋的**多智能体系统**世界，探讨 **Agent-to-Agent (A2A) 协议**，并探索如何使它们**安全有效地进行通信**。

> 💡 实用实施指南 在本白皮书中，所有实用示例均引用了 Google Cloud Platform Agent Starter Pack1——这是一个为 Google Cloud 提供生产就绪型生成式 AI 代理模板的 Python 包。它包括预构建的代理、自动化的 CI/CD 设置、Terraform 部署、Vertex AI 评估集成以及内置的 Google Cloud 可观测性。该入门包通过您可以在几分钟内部署的工作代码，演示了此处讨论的概念。

人员和流程

在谈论了 CI/CD、可观测性和动态管道之后，为什么还要关注人员和流程呢？因为世界上最好的技术，如果没有合适的团队来构建、管理和治理它，也是无效的。

那个客服**智能体**不是奇迹般地被阻止赠送免费产品；而是一位 **AI 工程师**和一位 **Prompt 工程师**设计并实施了防护栏。机密数据库不是通过一个抽象概念来保护的；而是一个**云平台团队**配置了身份验证。在每一个成功的、**生产级智能体**的背后，都有一个协调良好的专家团队，在本节中，我们将介绍这些关键角色。

实用实施指南 在本白皮书的始终，实际示例引用了 Google Cloud Platform Agent Starter Pack——这是一个 Python 软件包，为 Google Cloud 提供了**生产就绪**的**生成式 AI 智能体模板**。它包括预构建的**智能体**、自动化 CI/CD 设置、Terraform 部署、Vertex AI 评估集成 和内置的 Google Cloud 可观测性。该入门包用可在几分钟内部署的运行代码演示了本文讨论的概念。

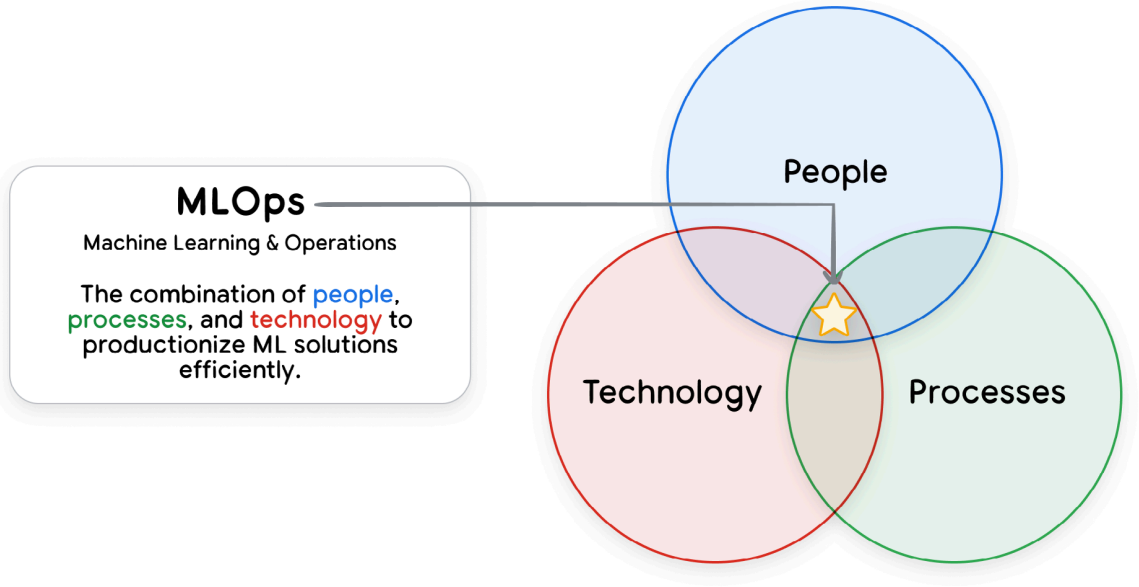


图 1：展示“运营 (Ops)”是人员、流程和技术的交集的图表

在传统的 MLOps 环境中，这涉及几个关键团队：

- **云平台团队：** 由云架构师、管理员和安全专家组成，该团队管理基础云基础设施、安全和访问控制。该团队为工程师和服务账户授予**最小权限角色**，确保他们只访问必要的资源。
- **数据工程团队：** 数据工程师和数据所有者构建和维护数据管道，负责数据摄取、准备和质量标准。
- **数据科学和 MLOps 团队：** 这包括进行模型实验和训练的数据科学家，以及使用 CI/CD 在规模上自动化**端到端 ML 管道**（例如，预处理、训练、后处理）的 ML 工程师。MLOps 工程师通过构建和维护标准化管道基础设施来支持这一点。
- **机器学习治理：** 这一集中式职能，包括产品负责人和审计师，负责监督 ML 生命周期，作为工件和指标的存储库，以确保**合规性、透明度和问责制**。

生成式 AI 为这一环境引入了**新的复杂性**和**专业角色**：

- **Prompt 工程师：** 尽管这个角色的名称在行业中仍在演变，但这些人员将**制作 Prompt 的技术技能**与**深厚的领域专业知识**相结合。他们定义了模型应回答的正确问题和预期答案，尽管在实践中，这项工作可能由 **AI 工程师、领域专家或专职专家**根据组织的成熟度来完成。
- **AI 工程师：** 他们负责将 GenAI 解决方案扩展到生产环境，构建强大的后端系统，该系统需要纳入**规模化评估、防护栏和 RAG/工具集成**。

- **DevOps/应用开发人员：** 这些开发人员构建**前端组件**和**用户友好的界面**，以与 GenAI 后端集成。

组织的规模和结构将影响这些角色；在较小的公司中，个人可能身兼数职，而成熟的组织将拥有**更专业的团队**。有效协调所有这些不同的角色，对于建立**稳健的运营基础** 并成功地将传统 ML 和**生成式 AI 计划投入生产** 至关重要。

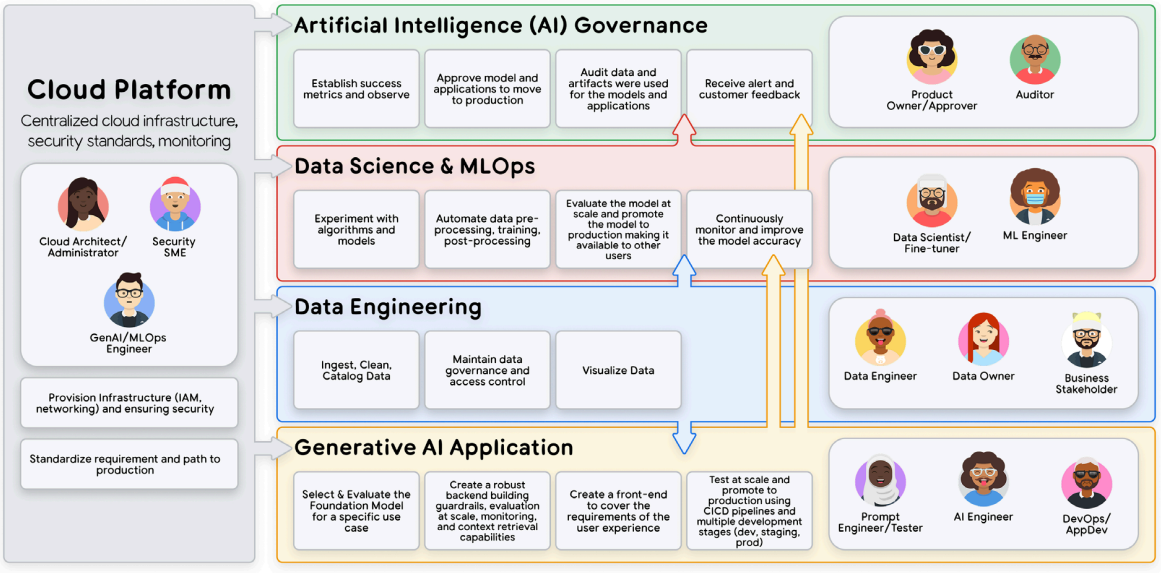


图 2：多个团队如何协作以实现模型与 GenAI（生成式 AI）应用的运营落地

迈向生产环境的旅程

现在我们已经确定了团队，接下来我们转向流程。我们如何将所有这些专家的工作转化为一个值得信任、可靠、并为用户做好准备的系统？

答案在于一个以单一核心原则为基础的规范的投产前流程：**评估门控部署 (Evaluation-Gated Deployment)**。这个想法简单而有力：任何**智能体**版本在通过全面的评估，证明其质量和安全性 之前，都不应接触到用户。在这个投产前阶段，我们将手动的不确定性换成自动化的信心，它由三个支柱组成：一个充当**质量关卡**的**严格评估流程**，一个强制执行它的**自动化 CI/CD 管道**，以及旨在降低进入生产环境最后一步风险的**安全部署策略**。

作为质量关卡的评估

*为什么我们需要为**智能体**设置一个特殊的质量关卡？* 传统的软件测试对于那些会推理和适应的系统来说是不够的。此外，评估一个**智能体**不同于评估一个 LLM；它不仅需

要评估最终答案，还需要评估完成任务所采取的**整个推理和行动轨迹**。一个**智能体**可能通过了 100 个工具的单元测试，但仍可能因为选择了错误的工具或产生幻觉响应而彻底失败。我们需要评估它的**行为质量**，而不仅仅是它的功能正确性。这个关卡可以通过两种主要方式实现：

1. **手动“预 PR”评估 (The Manual "Pre-PR" Evaluation)**：对于寻求灵活性或刚开始评估之旅的团队，质量关卡通过团队流程来强制执行。在提交拉取请求 (PR) 之前，**AI 工程师**或 **Prompt 工程师**（或组织中负责**智能体**行为的人）在本地运行评估套件。然后，将生成的**性能报告**——将新**智能体**与生产基线进行比较——链接到 PR 描述中。这使得评估结果成为**强制性的人工审查工件**。审查者——通常是另一位 **AI 工程师**或**机器学习治理人员**——现在负责评估代码，以及**智能体**针对**防护栏违规**和 **Prompt 注入漏洞**的行为变化。

2. **自动化管道内关卡 (The Automated In-Pipeline Gate)**：对于成熟的团队，**评估线束**——由数据科学和 MLOps 团队构建和维护——直接集成到 **CI/CD 管道**中。失败的评估会自动**阻止部署**，为**机器学习治理团队**定义的质量标准提供严格的、程序化的强制执行。这种方法用自动化的一致性换取了手动审查的灵活性。**CI/CD 管道**可以配置为自动触发一个评估作业，该作业将新**智能体**的响应与一个**黄金数据集**进行比较。如果“工具调用成功率”或“有用性”等关键指标低于预定义的阈值，部署将被程序化地阻止。

无论采用哪种方法，原则都是相同的：**没有经过质量检查的智能体不得进入生产环境**。我们在第 4 天的深入探讨中介绍了衡量什么以及如何构建这个评估线束的具体细节：《**智能体**质量：可观测性、日志、跟踪、评估、指标》，其中探讨了从制作**“黄金数据集”（一组精心策划的、具有代表性的测试用例，旨在评估**智能体**的预期行为和防护栏合规性）到实现“LLM-即-法官”技术，再到最终使用像 Vertex AI Evaluation2 这样的服务来驱动评估 的所有内容。

自动化 CI/CD 管道

一个 **AI 智能体**是一个**复合系统**，不仅包含源代码，还包含 **Prompt、工具定义和配置文件**。这种复杂性带来了重大挑战：我们如何确保对 Prompt 的更改不会降低工具的性能？在这些工件接触用户之前，我们如何测试它们之间的相互作用？

解决方案是 **CI/CD（持续集成/持续部署）管道**。它不仅仅是一个自动化脚本；它是一个结构化的流程，帮助团队中不同的人协作管理复杂性 并确保质量。它的工作原理是在阶段中测试更改，在**智能体**发布给用户之前逐步建立信心。

一个稳健的管道被设计成一个**漏斗**。它尽可能早、尽可能便宜地捕获错误，这种做法通常被称为**“左移”**。它将快速的、合并前的检查与更全面的、资源密集型的、合并后的部署分开。这种渐进式的工作流程通常分为**三个不同的阶段**：

1. **阶段 1：合并前集成 (CI)**。管道的首要职责是向发起拉取请求的 **AI 工程师**或 **Prompt 工程师**提供**快速反馈**。自动触发的 CI 阶段充当主分支的**守门人**。它运行快速检查，如**单元测试、代码 Linting 和依赖项扫描**。至关重要的是，这是运行 **Prompt 工程师**设计的**智能体质量评估套件**的理想阶段。这提供了关于更改是改进还是降低了**智能体**在关键场景中的性能的即时反馈，甚至在合并之前。通过在此处捕获问题，我们防止了污染主分支。使用 **Agent Starter Pack1 (ASP)** 生成的 PR 检查配置模板3 是使用 Cloud Build4 实施此阶段的实际示例。

2. **阶段 2：在暂存环境中进行合并后验证 (CD)**。一旦更改通过所有 CI 检查——包括性能评估——并被合并，焦点就从代码和性能正确性转移到集成系统的**操作准备情况**。持续部署 (CD) 流程（通常由 MLOps 团队管理）将**智能体**打包并部署到**暂存环境**——一个高保真度的生产副本。在这里，运行更全面、资源密集型的测试，例如**负载测试和针对远程服务的集成测试**。这也是**内部用户测试**（通常称为“Dogfooding”）的关键阶段，公司内部的人员可以与**智能体**交互，并在接触最终用户之前提供定性反馈。这确保了**智能体**作为一个集成系统在类似生产的条件下可靠且高效地运行，然后才考虑发布。ASP 的暂存部署模板5 展示了此部署的一个示例。

3. **阶段 3：受控部署到生产环境**。在暂存环境中对**智能体**进行彻底验证之后，最后一步是部署到生产环境。这几乎从不完全自动化，通常需要**产品负责人给出最终批准**，确保**有人工干预**。批准后，在暂存环境中测试和验证的精确部署工件将被提升到生产环境。ASP 生成的此生产部署模板6 展示了此最终阶段如何检索经过验证的工件 并将其部署到生产环境，同时带有适当的保护措施。

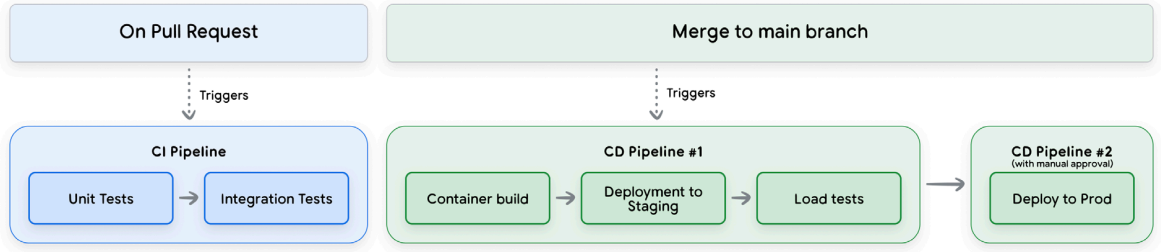


图 3：CI/CD 流程的不同阶段

实现这个三阶段 **CI/CD** 工作流程需要稳健的自动化基础设施和适当的**机密管理**。这种自动化由两个关键技术驱动：

- **基础设施即代码 (IaC)**：像 Terraform 这样的工具以编程方式定义环境，确保它们是相同的、可重复的、并且经过版本控制。例如，使用 Agent Starter Pack7 生成的此模板提供了完整的**智能体**基础设施的 Terraform 配置，包括 Vertex AI、Cloud Run 和 BigQuery 资源。

- **自动化测试框架**：像 Pytest 这样的框架在每个阶段执行测试和评估，处理**智能体**特定的工件，如**对话历史、工具调用日志和动态推理痕迹**。

此外，敏感信息，例如工具的 API 密钥，应使用 Secret Manager8 这样的服务进行**安全管理**，并在运行时注入到**智能体**的环境中，而不是硬编码在存储库中。

安全部署策略

虽然全面的投产前检查至关重要，但实际应用中不可避免地会暴露出不可预见的问题。与其一次性切换 100% 的用户，不如考虑通过渐进式部署和仔细监控来最小化风险。

以下是四种被证明有助于团队建立部署信心的模式：

- **金丝雀发布 (Canary)**：从 1% 的用户开始，监控 Prompt 注入和意外的工具使用。逐渐扩大规模或立即回滚。
- **蓝绿部署 (Blue-Green)**：运行两个相同的生产环境。将流量路由到“蓝色”环境，同时部署到“绿色”环境，然后立即切换。如果出现问题，立即切换回来——零停机时间，即时恢复。
- **A/B 测试**：比较不同**智能体**版本在真实业务指标上的表现，以实现数据驱动的决策。这可以与内部或外部用户一起发生。
- **特征标志 (Feature Flags)**：部署代码，但动态控制发布，首先与选定的用户测试新功能。

所有这些策略都有一个共同的基础：**严格的版本控制**。每个组件——代码、Prompt、模型端点、工具模式、内存结构，甚至评估数据集——都必须进行版本控制。当尽管有保护措施仍出现问题时，这能够**即时回滚到已知良好的状态**。将此视为你的生产环境“撤销”按钮！

你可以使用 Agent Engine9 或 Cloud Run10 部署**智能体**，然后利用 Cloud Load Balancing11 来管理跨版本的流量 或连接到其他微服务。Agent Starter Pack1 提供了带有 **GitOps 工作流程**的即用型模板——其中每次部署都是一个 Git Commit，每次回滚都是一个 Git Revert，你的存储库成为当前状态和完整部署历史的**唯一真相来源**。

从一开始就构建安全性

安全的部署策略可以保护你免受 Bug 和中断的影响，但**智能体**面临一个独特的挑战：它们可以自主推理和行动。一个完美部署的**智能体**如果未内置适当的安全和责任措施，仍然可能造成伤害。这需要一个从第一天就嵌入，而不是作为事后补救措施添加的**综合治理策略**。

与遵循预定路径的传统软件不同，**智能体**会做出决策。它们解释模糊的请求，访问多个工具，并跨会话维护内存。这种自主性带来了独特的风险：

- **Prompt 注入和恶意行为**：恶意用户可以欺骗**智能体**执行意外操作或绕过限制。
- **数据泄露**：**智能体**可能通过其响应或工具使用不经意地暴露敏感信息。
- **内存污染**：存储在**智能体**内存中的虚假信息可能会破坏所有未来的交互。

幸运的是，像 Google 的 Secure AI Agents 方法12 和 Google Secure AI Framework (SAIF)13 这样的框架通过**三层防御**来应对这些挑战：

1. **政策定义和系统指令（智能体的宪法）**：流程始于定义**智能体**期望和不期望行为的政策。这些政策被工程化为**系统指令 (SIs)**，作为**智能体**的**核心宪法**。

2. **防护栏、保障措施和过滤（强制执行层）**：这一层充当**硬停止强制执行机制**。

◦ **输入过滤**：使用分类器和像 Perspective API 这样的服务来分析 Prompt，并在恶意输入到达**智能体**之前阻止它们。

◦ **输出过滤**：在**智能体**生成响应后，Vertex AI 的内置安全过滤器提供了一个最终检查，用于检查有害内容、PII 或政策违规。例如，在响应发送给用户之前，它会通过 Vertex AI 的内置安全过滤器14，该过滤器可以配置为阻止包含特定 PII、有毒语言或其他有害内容的输出。

◦ **人工干预 (HITL) 升级**：对于高风险或模糊的操作，系统必须暂停并升级给人工进行审查和批准。

3. **持续保障和测试**：安全不是一次性的设置。它需要**持续的评估和适应**。

◦ **严格评估**：对模型或其安全系统的任何更改都必须触发使用 Vertex AI Evaluation 进行全面重新运行的评估管道。

◦ **专用 RAI 测试**：通过创建专用数据集或使用模拟**智能体**，严格测试特定风险，包括中立观点 (NPOV) 评估和平等评估。

◦ **主动红队测试 (Proactive Red Teaming)**：通过创造性的手动测试和 AI 驱动的基于角色的模拟，积极尝试破坏安全系统。

生产环境中的运营

你的**智能体**已上线。现在焦点从开发转移到一个根本不同的挑战：在系统与数千用户交互时，保持其**可靠性、成本效益和安全性**。一个传统服务基于可预测的逻辑运行。一个**智能体**，相比之下，是一个**自主的行动者**。它遵循意外推理路径的能力意味着它可以在没有直接监督的情况下表现出**涌现行为**和**累积成本**。

管理这种自主性需要不同的运营模型。有效的团队不再采用静态监控，而是采用一个**连续的循环**：**观察**系统的实时行为，**行动**以维持性能和安全，并根据生产环境中的学习来**演进智能体**。这种集成循环是在生产环境中成功运营**智能体**的**核心准则**。

观察：你的智能体的感官系统

要信任和管理一个**自主智能体**，你必须首先了解它的流程。**可观测性**提供了这种至关重要的洞察，充当后续“行动”和“演进”阶段的**感官系统**。稳健的**可观测性**实践建立在**三个支柱**之上，它们协同工作以提供**智能体**行为的完整图景：

- **日志 (Logs)**：发生了什么的细粒度、事实性日记，记录了每一次工具调用、错误和决策。
- **跟踪 (Traces)**：连接单个日志的叙事，揭示**智能体**采取某个行动的**原因路径**。
- **指标 (Metrics)**：聚合的报告卡，总结了**性能、成本和操作健康状况**，以显示系统在大规模上运行得有多好。

例如，在 Google Cloud 中，这是通过运营套件实现的：用户的请求在 **Cloud Trace** 中生成一个唯一 ID，该 ID 将 Vertex AI Agent Engine 调用、模型调用和工具执行与可见的持续时间连接起来。详细日志流向 **Cloud Logging**，而 **Cloud Monitoring** 仪表板在延迟阈值超过时发出警报。**Agent Development Kit (ADK)** 为**智能体**操作的**自动插桩**提供了内置的 **Cloud Trace** 集成。

通过实施这些支柱，我们从在黑暗中操作转向对**智能体**的行为有了清晰的、数据驱动的视图，为在生产环境中有效管理它提供了所需的基础。

行动：运营控制的杠杆

没有行动的观察只是**昂贵的仪表板**。“行动 (Act)”阶段是关于**实时干预**——你根据观察到的情况来拉动的杠杆，以管理**智能体**的性能、成本和安全。

将“行动”视为系统的**自动化反射**，旨在实时维护稳定性。相比之下，稍后将介绍的“演进 (Evolve)”是学习行为以创建一个**根本上更好的系统**的战略过程。

因为**智能体是自主的**，你无法预先编程所有可能的结果。相反，你必须构建**稳健的机制**来影响它在生产环境中的行为。这些**运营杠杆**分为两个主要类别：**管理系统健康**和**管理系统风险**。

管理系统健康：性能、成本和规模

与传统的微服务不同，**智能体**的工作负载是**动态且有状态**的。管理其健康状况需要处理这种不可预测性的策略。

- **设计可扩展性**：基础是将**智能体**的**逻辑与其状态解耦**。

◦ **水平扩展**：将**智能体**设计为**无状态、容器化的服务**。有了外部状态，任何实例都可以处理任何请求，从而使像 **Cloud Run** 或托管的 **Vertex AI Agent Engine Runtime** 这样的无服务器平台能够**自动扩展**。

- **异步处理**：对于长时间运行的任务，使用**事件驱动的模式来卸载工作**。这使得**智能体**保持响应性，而复杂的作业在后台处理。例如，在 Google Cloud 上，一个服务可以向 **Pub/Sub** 发布任务，然后 **Pub/Sub** 可以触发一个 **Cloud Run** 服务进行异步处理。
- **外部化状态管理**：由于 LLM 是无状态的，将内存**外部化持久化**是不可协商的。这突出一个关键的架构选择：**Vertex AI Agent Engine** 提供了内置的、持久的会话和内存服务，而 **Cloud Run** 则提供了直接与像 **AlloyDB** 或 **Cloud SQL** 这样的数据库集成的灵活性。
- **平衡竞争目标**：扩展总是涉及平衡**三个竞争目标：速度、可靠性和成本**。

◦ **速度（延迟）**：通过将其设计为**并行工作**、**积极缓存结果**，以及对常规任务使用**更小、更高效的模型**，来保持**智能体**的快速。

◦ **可靠性（处理故障）**：**智能体**必须处理临时故障。当调用失败时，**自动重试**，最好采用**指数退避**，以便给服务时间恢复。这需要设计“可安全重试”（**幂等**）的工具，以防止像重复收费这样的错误。

◦ **成本**：通过**缩短 Prompt**、对更简单的任务使用**更便宜的模型**，以及**分组发送请求（批处理）** 来保持**智能体**的经济性。

管理风险：安全响应手册

因为**智能体可以自行行动**，你需要一个**快速遏制的手册**。当检测到威胁时，响应应遵循清晰的顺序：**遏制、分流和解决**。

第一步是**立即遏制**。首要任务是**阻止伤害**，通常使用“**断路器**”——一个用于**立即禁用受影响工具的特征标志**。

接下来是**分流**。在威胁被遏制后，可疑请求被路由到**人工干预 (HITL) 审查队列**，以调查漏洞的范围和影响。

最后，重点转向**永久解决**。团队开发一个补丁——例如**更新的输入过滤器或系统 Prompt**——并通过**自动化 CI/CD 管道部署它**，确保修复在彻底测试后永久阻止漏洞。

演进：从生产环境中学习

虽然“行动”阶段提供了系统的即时、战术性反射，但“演进 (Evolve)”阶段是关于**长期、战略性的改进**。它始于查看在可观测性数据中收集的模式和趋势，并提出一个关

键问题："我们如何修复**根本原因**，让这个问题永远不会再发生？"

在这里，你将从对生产事件的反应 转向**主动使你的智能体更智能、更高效、更安全**。你将"观察"阶段的原始数据转化为**智能体**架构、逻辑和行为的**持久改进**。

演进的引擎：自动化通往生产环境的路径

来自生产环境的洞察只有在你能够迅速采取行动时才有价值。观察到 30% 的用户在特定任务上失败是没用的，如果你的团队需要六个月才能部署修复。

这就是你在投产前构建的**自动化 CI/CD 管道**（第 3 节） 成为你运营循环中**最关键的组成部分**。它是推动**快速演进的引擎**。一条快速、可靠的通往生产环境的路径，使你能够在**数小时或数天内**，而不是数周或数月内，关闭观察和改进之间的循环。

当你确定了一个潜在的改进——无论是**完善的 Prompt、新工具，还是更新的安全防护栏**——流程应该是：

1. **提交更改**：提议的改进被提交到你的**版本控制存储库**。

2. **触发自动化**：提交自动触发你的 **CI/CD 管道**。

3. **严格验证**：管道针对你更新的数据集运行**全套单元测试、安全扫描和智能体质量评估套件**。

4. **安全部署**：一旦验证，更改将使用**安全部署策略部署到生产环境**。

这种自动化工作流程将演进从一个缓慢、高风险的手动项目，转变为一个**快速、可重复和数据驱动的流程**。

演进工作流：从洞察到部署改进

1. **分析生产数据**：从生产日志中识别用户行为、任务成功率和安全事件的趋势。

2. **更新评估数据集**：将生产故障转化为明天的测试用例，扩充你的**黄金数据集**。

3. **完善和部署**：提交改进以触发自动化管道——无论是**完善 Prompt、添加工具，还是更新防护栏**。

这创建了一个**良性循环**，你的**智能体**在每一次用户交互中持续改进。

💡 **演进循环的实际应用** 一个零售**智能体**的日志（观察）显示，15% 的用户在询问“类似产品”时收到错误。产品团队采取行动（Act），创建了一个高优先级工单。演进阶段（Evolve）开始：生产日志被用于为评估数据集创建一个新的、失败的测试用例。一位 **AI 工程师**完善了**智能体**的 Prompt，并添加了一个新的、更强大的**相似性搜索工具**。更改被提交，通过了 **CI/CD 管道**中现在已更新的**评估套件**，并通过**金丝雀部署安全地发布**，在 **48 小时内**解决了用户问题。

演进安全性：生产反馈循环

尽管基础的安全和责任框架是在投产前建立的（第 3.4 节），但工作从未真正完成。**安全不是一个静态的清单**；它是一个**动态的、持续的适应过程**。生产环境是**最终的试验场**，在那里收集的洞察对于使你的**智能体**抵御**现实世界威胁**至关重要。

这就是 **观察 → 行动 → 演进** 循环对**安全至关重要**的地方。这个过程是演进工作流程的**直接延伸**：

1. **观察**：你的监控和日志系统检测到**新的威胁载体**。这可能是一种**绕过当前过滤器的新颖 Prompt 注入技术**，或者导致轻微数据泄露的意外交互。

2. **行动**：立即的安全响应团队**遏制威胁**（如第 4.2 节所讨论）。

3. **演进**：这是**长期恢复力**的关键一步。安全洞察被反馈回你的开发生命周期：

- **更新评估数据集**：新的 Prompt 注入攻击作为**永久测试用例**被添加到你的**评估套件**中。

- **完善防护栏**：**Prompt 工程师**或 **AI 工程师**完善**智能体**的**系统 Prompt、输入过滤器或工具使用策略**，以阻止新的攻击载体。

- **自动化和部署**：工程师提交更改，这会**触发完整的 CI/CD 管道**。更新后的**智能体**经过针对**新扩展评估集**的**严格验证**，并**部署到生产环境**，从而关闭漏洞。

这创造了一个**强大的反馈循环**，每一次生产事件都使你的**智能体更强大、更有弹性**，将你的安全态势从防御立场转变为**持续、主动的改进**。

要了解更多关于**负责任的 AI 和保护 AI 智能体系统**的信息，请查阅白皮书 **Google 的 Secure AI Agents 方法** 和 **Google Secure AI Framework (SAIF)**。

超越单智能体运营

你已经掌握了在生产环境中运营**单个智能体**，并能以高速度交付它们。但是，随着组织扩展到数十个**专业智能体**——每个**智能体**都由不同的团队使用不同的框架构建——一个新的挑战出现了：这些**智能体**无法协作。下一节将探讨标准化协议如何将这些孤立的**智能体**转变为一个**可互操作的生态系统**，通过**智能体协作**释放**指数级的价值**。

A2A - 可重用性与标准化

你的组织内部已经构建了数十个**专业智能体**。客户服务团队有他们的支持**智能体**。分析团队构建了预测系统。风险管理团队创建了欺诈检测。但问题在于：这些**智能体**之间无法相互通信——无论是由于它们是用不同的框架、项目还是完全不同的云环境创建的。

这种**隔离造成了巨大的低效**。每个团队都重复构建相同的功能。关键洞察仍被困在孤岛中。你需要的是**互操作性**——即任何**智能体**都能够利用任何其他**智能体**的能力，无论它是谁构建的或使用了什么框架。

要解决这个问题，需要一种基于**两种不同但互补协议**的**规范化方法**。虽然**模型上下文协议 (MCP)**（我们在《**智能体**工具和与 MCP 的互操作性》中详细介绍过）为**工具集成**提供了一个通用标准，但它不足以满足**智能智能体**之间所需的**复杂、有状态的协作**。这就是 **Agent2Agent (A2A) 协议**（现由 Linux 基金会治理）旨在解决的问题。

这种区别至关重要。当你需要一个简单的、**无状态的功能**，例如获取天气数据或查询数据库时，你需要一个会说 **MCP** 的工具。但是，当你需要**委派一个复杂的目标**，例如“分析上一季度的客户流失并推荐三种干预策略”时，你需要一个能够通过 **A2A** 进行**推理、规划和自主行动的智能伙伴**。简而言之，**MCP** 让你说“做这个特定的事情”，而 **A2A** 让你说“实现这个复杂的目标”。

A2A 协议：从概念到实现

A2A 协议旨在打破组织孤岛，实现**智能体**之间的**无缝协作**。考虑一个场景，一个欺诈检测**智能体**发现了可疑活动。为了理解完整的上下文，它需要来自一个单独的交易分析**智能体**的数据。如果没有 A2A，人类分析师必须手动弥合这一差距——这个过程可能需要数小时。有了 A2A，**智能体**会自动协作，在几分钟内解决问题。

协作的第一步是发现正确的委派**智能体**——这通过 **Agent Cards**24 使其成为可能，**Agent Cards** 是充当每个**智能体**名片的**标准化 JSON 规范**。**Agent Card** 描述了一个**智能体**可以做什么、它的安全要求、它的技能 以及如何联系它 (url)，允许生态系统中的任何其他**智能体****动态地发现其对等体**。请参阅下面的 **Agent Card** 示例：

代码片段 1：一个用于 `check_prime_agent` 的示例**智能体卡**

```
{
"name": "check_prime_agent",
"version": "1.0.0",
"description": "An agent specialized in checking whether numbers are prim
e",
"capabilities": {},
"securitySchemes": {
"agent_oauth_2_0": {
"type": "oauth2",
}
"defaultInputModes": ["text/plain"],
"defaultOutputModes": ["application/json"],
"skills": [
{
"id": "prime_checking",
"name": "Prime Number Checking",
"description": "Check if numbers are prime using efficient algorithms",
"tags": ["mathematical", "computation", "prime"]
}
],
"url": "http://localhost:8001/a2a/check_prime_agent"
}
```

采用此协议不需要架构大修。像 **ADK** 这样的框架显著简化了这一过程（文档25）。你可以通过一个函数调用使现有**智能体**兼容 A2A，该调用会自动生成其 **AgentCard**，并使其在网络上可用。

代码片段 2：使用 **ADK** 的 `to_a2a` 实用程序包装现有**智能体**并将其暴露给 A2A 通信

```
# Example using ADK: Exposing an agent via A2A
from google.adk.a2a.utils.agent_to_a2a import to_a2a

# Your existing agent
root_agent = Agent(
    name='hello_world_agent',
    # ... your agent code ...
)
# Make it A2A-compatible
```

```
a2a_app = to_a2a(root_agent, port=8001)

# Serve with uvicorn
# uvicorn agent:a2a_app --host localhost --port 8001
# Or serve with Agent Engine
# from vertexai.preview.reasoning_engines import A2aAgent
# from google.adk.a2a.executor.a2a_agent_executor import A2aAgentExec
utor
# a2a_agent = A2aAgent(
#    agent_executor_builder=lambda: A2aAgentExecutor(agent=root_agent)
# )
```

一旦一个**智能体**暴露，任何其他**智能体**都可以通过引用其 **AgentCard** 来使用它。例如，一个客服**智能体**现在可以查询一个**远程产品目录智能体**，而无需了解其内部工作原理。

代码片段 3：使用 **ADK** 的 `RemoteA2aAgent` 类连接和使用远程**智能体**

```
# Example using ADK: Consuming a remote agent via A2A
from google.adk.agents.remote_a2a_agent import RemoteA2aAgent

prime_agent = RemoteA2aAgent(
    name="prime_agent",
    description="Agent that handles checking if numbers are prime.",
    agent_card="http://localhost:8001/a2a/check_prime_agent/
            .well-known/agent-card.json"
)
```

这解锁了**强大的、层次化的组合**。一个根**智能体**可以配置为同时编排用于简单任务的本地子**智能体**和通过 A2A 的远程专业**智能体**，从而创建一个能力更强的系统。

代码片段 4：在 **ADK** 中使用远程 A2A **智能体** (`prime_agent`) 作为**层次化智能体结构**中的子**智能体**

```
# Example using ADK: Hierarchical agent composition
# ADK Local sub-agent for dice rolling
roll_agent = Agent(
    name="roll_agent",
    instruction="You are an expert at rolling dice."
)
```

```
# ADK Remote A2A agent for prime checking
prime_agent = RemoteA2aAgent(
    name="prime_agent",
    agent_card="http://localhost:8001/.well-known/agent-card.json"
)

# ADK Root orchestrator combining both
root_agent = Agent(
    name="root_agent",
    instruction="""Delegate rolling dice to roll_agent, prime checking to prim
e_agent.""",
    sub_agents=[roll_agent, prime_agent]
)
```

然而，启用这种程度的**自主协作**引入了两个**不可协商的技术要求**。首先是**分布式跟踪**，其中每个请求都携带一个**唯一的跟踪 ID**，这对于跨多个**智能体**的调试和维护一致的**审计跟踪**至关重要。其次是**稳健的状态管理**。A2A 交互本质上是**有状态的**，需要一个复杂的**持久层**来跟踪进度和确保**事务完整性**。

A2A 最适合需要**持久服务合同**的**正式、跨团队集成**。对于单个应用程序内紧密耦合的任务，**轻量级的本地子智能体**通常仍然是更有效的选择。随着生态系统的成熟，新的**智能体**的构建应**原生支持这两种协议**，确保每个新组件都**立即可发现、可互操作和可重用**，从而使整个系统的价值复合增长。

A2A 和 MCP 如何协同工作

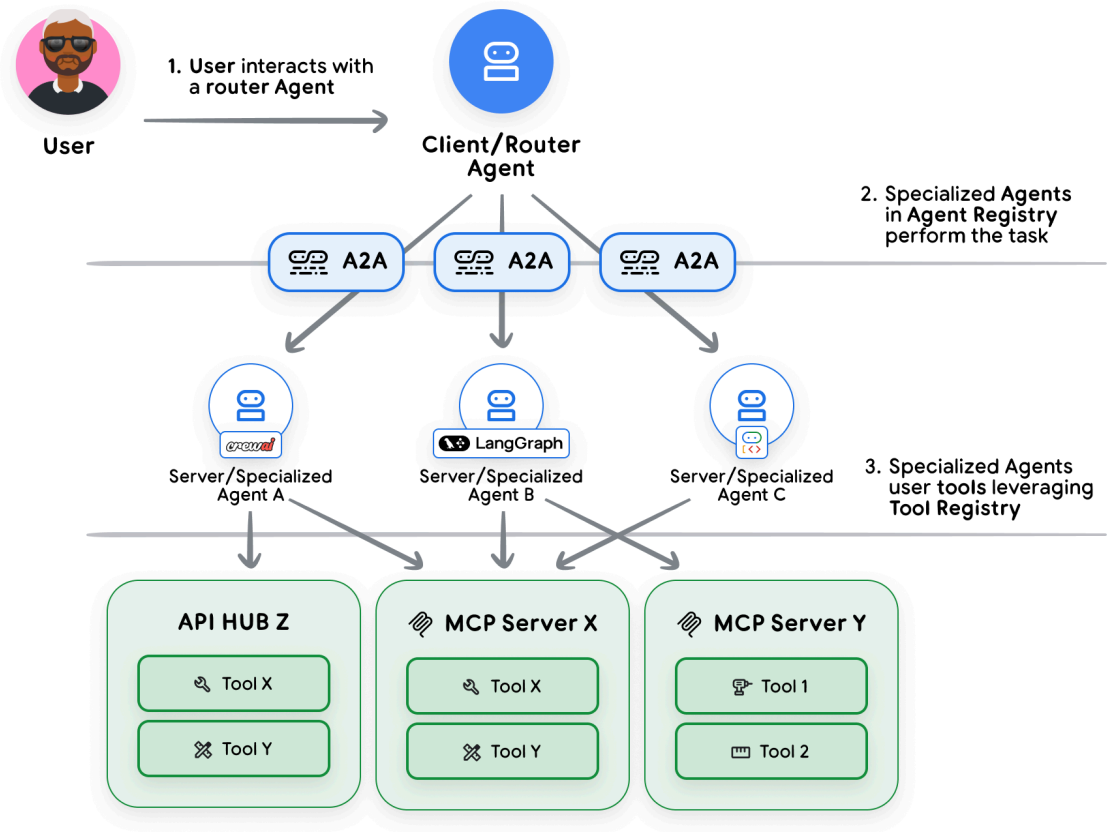


图 4：A2A 和 MCP 协作的一瞥

A2A 和 MCP 不是**竞争标准**；它们是设计用于在**不同抽象级别**操作的**互补协议**。区别取决于一个**智能体**正在与什么交互。**MCP** 是**工具和资源**的领域——具有明确定义的、**结构化输入和输出**的原语，像一个计算器或一个数据库 API。**A2A** 是**其他智能体**的领域——它们是可以**推理、规划、使用多个工具并维护状态**以实现**复杂目标**的**自主系统**。

最强大的**智能体系统**在**分层架构**中同时使用这两种协议。一个应用程序可能主要使用 **A2A** 来编排多个**智能智能体**之间**的高级协作**，而这些**智能体**中的每一个都在内部使用 **MCP** 来与其自己特定的**工具和资源集**进行交互。

一个实用的类比是一个由**自主 AI 智能体**组成**汽车修理店**。

1. **用户到智能体 (A2A)**：客户使用 **A2A** 与“店经理”**智能体**通信，描述一个**高级问题**：“我的车发出咔嗒声”。

2. **智能体到智能体 (A2A)**：店经理进行**多轮诊断对话**，然后使用 **A2A** 将任务委派给专业的“机械师”**智能体**。

3. **智能体到工具 (MCP)**：机械师**智能体**现在需要执行**特定的操作**。它使用 **MCP** 调用其**专业工具**：它在诊断扫描仪上运行 `scan_vehicle_for_error_codes()`，使用 `get_repair_procedure()` 查询维修手册数据库，并使用 `raise_platform()` 操作平台升降机。

4. **智能体到智能体 (A2A)**：在诊断问题后，机械师**智能体**确定需要一个零件。它使用 **A2A** 与**外部**的"零件供应商"**智能体**通信，询问可用性并下订单。

在这个工作流程中，**A2A** 促进了客户、商店的**智能体**和外部供应商之间**更高级别的、对话式的和面向任务的交互**。同时，**MCP** 提供了**标准化的管道**，使机械师**智能体**能够可靠地使用其**特定的、结构化的工具**来完成工作。

注册表架构：何时以及如何构建它们

为什么有些组织构建注册表而有些不需要？答案在于**规模和复杂性**。当你有五十个工具时，**手动配置**运行良好。但是，当你达到分布在不同团队和环境中的**五千个工具**时，你面临一个需要**系统解决方案**的**发现问题**。

工具注册表使用像 **MCP** 这样的协议来编目所有资产，从函数到 API。与其让**智能体**访问数千个工具，不如创建精选列表，从而产生**三种常见模式**：

- **通才智能体**：访问完整目录，以速度和准确性换取范围。
- **专家智能体**：使用预定义子集以获得更高性能。
- **动态智能体**：在运行时查询注册表以适应新工具。

主要好处是**人类发现**——开发人员可以在构建重复项之前搜索现有工具，安全团队可以**审计工具访问**，产品负责人可以了解其**智能体**的能力。

智能体注册表将相同的概念应用于**智能体**，使用像 **A2A 的 AgentCards** 这样的格式。它帮助团队**发现和重用现有智能体**，减少冗余工作。这也为**自动化的智能体到智能体委派**奠定了基础，尽管这仍然是一种**新兴模式**。注册表以**维护成本**提供**发现和治理**。你可以考虑从**不构建注册表**开始，只有当你的**生态系统规模需要集中管理**时才构建它。

注册表的决策框架

- **工具注册表**：当**工具发现成为瓶颈** 或**安全需要集中审计**时构建。
- **智能体注册表**：当**多个团队需要在不紧密耦合的情况下发现和重用专业智能体**时构建

整合所有要素：AgentOps 生命周期

我们现在可以将这些支柱整合为一个单一的、有凝聚力的**参考架构**！生命周期始于开发人员的**内部循环**——这是一个快速进行**本地测试和原型设计**，以塑造**智能体**核心逻辑的阶段。一旦更改准备就绪，它就进入正式的**投产前引擎**，其中**自动化评估关卡**根据**黄金数据集**验证其质量和安全性。从那里，**安全部署**将其发布到生产环境，其中**全面的可观测性**捕获驱动**持续演进循环**所需的真实世界数据，将每一个洞察转化为下一次改进。

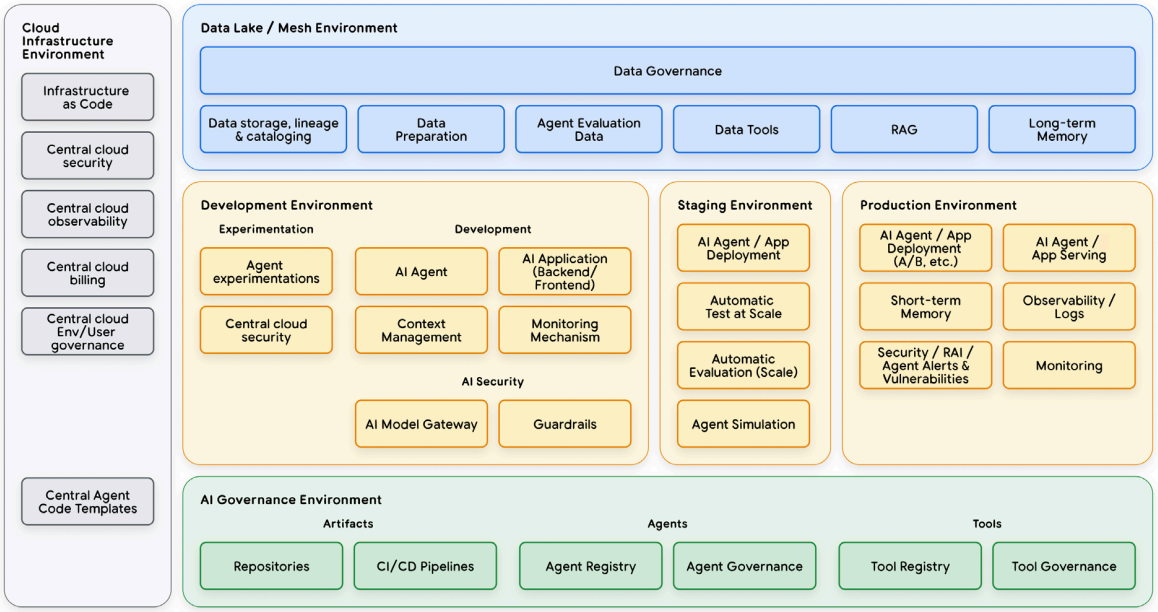


图 5：AgentOps 核心能力、环境和流程

结论：用 AgentOps 弥合最后一公里差距

将 AI 原型转移到生产系统是一场组织转型，需要一种新的运营准则：**AgentOps**。

大多数**智能体**项目失败在“**最后一公里**”，不是因为技术，而是因为**自主系统的运营复杂性被低估了**。本指南规划了弥合这一差距的路径。它始于建立**人员和流程**作为治理的基础。接下来，一个建立在**评估门控部署**基础上的投产前策略，实现了高风险发布的自动化。一旦上线，一个持续的**观察 → 行动 → 演进循环**将每一次用户交互转化为潜在的洞察。最后，**互操作性协议**通过将孤立的**智能体**转化为一个**协作的、智能的生态系统**来扩展系统。

即时的好处——比如防止安全漏洞或实现快速回滚——证明了投资的合理性。但真正的价值在于**速度**。成熟的 **AgentOps 实践**允许团队在**数小时而不是数周内部署改进**，

将静态部署转变为**持续演进的产品**。

你的前进之路

- 如果你刚开始，请专注于基础：构建你的**第一个评估数据集**，实施 **CI/CD 管道**，并建立全面的监控。Agent Starter Pack 是一个很好的起点——它可以在几分钟内创建一个具备这些基础的生产就绪**智能体**项目。
- 如果你正在扩展，请提升你的实践：**自动化从生产洞察到部署改进的反馈循环**，并标准化**互操作协议**，以构建一个有凝聚力的生态系统，而不仅仅是点对点解决方案。

下一个前沿不仅仅是构建更好的个体**智能体**，而是编排复杂的、学习和协作的**多智能体系统**。**AgentOps** 的运营准则就是实现这一目标的基础。我们希望这本手册能够赋能你构建下一代**智能、可靠和值得信赖的 AI**。弥合**最后一公里差距**因此并非项目的最后一步，而是**创造价值的第一步**！