What is AgentOps?

AgentOps—short for agent operations—is an emerging set of practices focused on the lifecycle management of autonomous AI agents. AgentOps brings together principles from previous operational disciplines like DevOps and MLOps, giving practitioners better methods to manage, monitor and improve agentic development pipelines.

Estimated at around USD 5 billion in 2024, the AI agents market is projected to grow to about USD 50 billion by 2030.[1] Yet as more enterprises build AI agents to streamline and automate workflows, new challenges emerge in monitoring the behavior of those agents, ensuring they perform as intended. AgentOps is a roughly-defined set of emerging best practices in evaluating agent performance, which builds on precepts established in the related fields of DevOps (which standardized software delivery) and MLOps (which did the same for machine learning models).

But managing agents isn't as straightforward as building traditional software or even AI models. "Agentic" systems are complex and dynamic, essentially involving software with a mind of its own. Agents act autonomously, chain tasks, make decisions and behave non-deterministically. The idea behind AgentOps is to bring observability and reliability into a realm that could be chaotic, enabling developers to peer into the black box of agent interactions and other agent behavior.

There is no single tool to manage AgentOps, but rather an entire ecosystem; a recent study discovered 17 tools on Github and other code repositories relevant to the practice, from Agenta to LangSmith to Trulens (One ambitiously named AgentOps tool is called, simply, "AgentOps"). These tools typically provide support to developers' agent framework of choice, be it IBM's watsonx Agents or OpenAI's Agents SDK. In this heated space, many popular platforms and frameworks have emerged, including AutoGen, LangChain and CrewAI (the latter optimized for the orchestration of multi-agent systems).

Your subscription will be delivered in English. You will find an unsubscribe link in every newsletter. You can manage your subscriptions or unsubscribe here. Refer to our IBM Privacy Statement for more information.

Why is AgentOps important?

An AI agent built to handle customer support tickets, for example, is likely comprised of one or more large language models (LLMs) using various tools to handle various tasks. Its agent workflow might involve monitoring incoming emails, searching a company knowledge base, and autonomously creating support tickets.

Debugging such an agent is complex; its varied behavior creates multiple points of potential failure or inefficiency. With agent monitoring, though, developers can conduct step-by-step session replays of agent runs, observing what the AI system did and when. Did the agent refer to the proper customer support documentation? What were the tool usage patterns, and just which APIs were used? What was the latency of each step? What was the ultimate LLM cost? How well did the agent communicate or collaborate with others?

Turning loose an AI agent without a plan to audit its behavior is something like giving a teenager a credit card and not looking at the resulting statement. Adam Silverman, the COO of Agency AI, recently told the Google for Developers blog that by using different LLMs for different tasks, that cost could be reduced—one of the many parameters that can be tweaked to optimize an agent's cost-effectiveness over time.[2]

Drilling deeper, developers can trace the agent's end-to-end behavior, including the cost of each LLM interaction across different providers (like Azure or AWS). Developers can consult a dashboard of such metrics in real time, with data from the various stages of the agent's lifecycle. Through iterative benchmarking, developers can then work towards the optimization of their agent.

Approaches to AgentOps

There is no universally agreed upon means of conducting AgentOps, with multiple tools and approaches available. (Indeed, even the much more established precursor term, DevOps, means slightly different things to different people). In June, at the IBM Think conference, IBM Research unveiled its own approach to AgentOps, specifying three core focus areas it believes are crucial to support observability with enterprise agentic AI use cases.

First, IBM Research built its AgentOps solution on top of OpenTelemetry (OTEL) standards, an open-source software development kit (SDK), allowing both automatic and manual instrumentations across various agentic frameworks. Second, it built an open analytics platform atop OTEL, giving users a high level of resolution when peering under the hood at their agents' behavior. The platform is extensible, meaning new metrics can easily be added. And third, these analytics are themselves powered by AI, enabling unique perspectives including multi-trace workflow views and trajectory explorations.

IBM Research used its AgentOps approach to assist the building of several IBM automation products, including Instana, Concert and Apptio. As IBM has brought its own agentic solutions to market, aspects of AgentOps have become features in the watsonx.ai developer studio and watsonx.governance toolkit for scaling trusted AI.

There are many approaches to AgentOps however, and the field is quickly evolving to meet the needs of an industry adopting agentic workflows at a dizzying speed.

Functions of AgentOps

The best practices of AgentOps can and should be applied to all phases of an agent's lifecycle.

Development: In this phase, developers give their agents specific objectives and constraints, mapping out various dependencies and data pipelines.

Testing: Before being released into a production environment, developers can evaluate how the agent performs in a simulated “sandbox” environment.

Monitoring: Once deployed, developers can examine the results of their instrumentation, evaluating agent performance on the level of the session, trace, or span. Developers can review agent actions, API calls and overall duration (or latency) of agent behavior.

Feedback: In this phase, both the user and developer need access to tooling to register when the agent made a mistake or behaved inconsistently, as well as mechanisms to help the agent perform better on its next run.

Governance: As generative AI comes under more regulatory scrutiny (as in the EU AI Act), and as new ethical frameworks evolve, developers need a set of guardrails and policies to help constrain agent behavior and ensure compliance.