Generative AI Lifecycle Operational Excellence framework on AWS

AWS Prescriptive Guidance

AWS Prescriptive Guidance: Generative AI Lifecycle Operational Excellence framework on AWS

Table of Contents

Generative AI Lifecycle Operational Excellence framework on AWS

Amazon Web Services (contributors)

November 2025 (document history)

Generative AI is revolutionizing industries with unprecedented opportunities for innovation and efficiency. However, the transition from experimental proof-of-concepts (PoCs) to production-grade systems presents unique operational challenges that surpass traditional software development tactics. The Generative AI Lifecycle Operational Excellence (GLOE) framework is intended to address these challenges through best practices and a staged approach. It guides organizations through the entire lifecycle of generative AI application development and operations.

GLOE addresses the complexities of large language models (LLMs) by providing suggestions to help you manage non-deterministic outputs, dynamic prompt evolution, and continuous adaptation needs in real-world scenarios. The framework combines component-based architectures, risk-based governance, and specialized operational practices while emphasizing strategic value. This multifaceted approach helps organizations transform generative AI uncertainties into predictable business outcomes.

Intended audience

This guide is intended for engineers, developers, data scientists, project managers, and business leaders who are developing generative AI applications. For more detailed information, see Personas who benefit from GLOE in this guide. To understand the concepts and recommendations in this guide, you should be familiar with the fundamentals of foundation models (FMs), including a basic understanding of LLMs. You should also be familiar with prompt engineering principles and have a basic understanding of AWS services, machine learning operations (MLOps) concepts, and DevOps concepts. This guide is intended for organizations that are starting generative AI PoCs or transitioning from experimental PoCs to production-grade generative AI systems.

Objectives

The recommendations in this guide can help you achieve the following:

- Convert prototypes into production-ready generative AI solutions that deliver measurable business value while reducing development time and costs.
- Implement a structured approach to develop generative AI applications and manage their lifecycles.
- Establish evaluation frameworks to handle non-deterministic AI outputs from generative AI applications.
- Build trust through proper validation mechanisms, ethical use practices, and control systems that promote compliance and responsible AI deployment.
- Monitor and maintain generative AI applications in production.
- Transform AI prototypes into scalable, production-ready systems.

Understanding the GLOE framework for generative AI

Generative AI Lifecycle Operational Excellence (GLOE) is a structured, iterative framework that manages the complete lifecycle of generative AI applications, from ideation to deployment and monitoring. Its core objectives are to standardize processes, enable team collaboration, and promote operational excellence throughout the generative AI application lifecycle. It is designed to help organizations deliver reliable, ethical, and adaptable AI solutions.

The framework transforms unstructured development practices into a systematic methodology that is designed for the long-term evolution of generative AI applications. GLOE's value lies in building resilient, future-proof generative AI operations that can adapt to change and consistently deliver value in a rapidly evolving AI landscape. This section describes the GLOE framework, including the challenges it addresses, its guiding principles, how it benefits different technical audiences, and a description of its stages.

This section contains the following topics:

Common generative AI challenges

The implementation of generative AI applications in enterprise environments presents a complex set of operational challenges that extend beyond conventional software development practices and traditional machine learning operations. The following common challenges collectively emphasize the necessity for the specialized GLOE framework:

- **Managing non-deterministic outputs** – LLMs can demonstrate impressive capabilities in controlled demonstrations, but they often exhibit unpredictable behavior in real-world scenarios. This inherent non-determinism makes debugging and validating consistent quality particularly difficult.
- **Dynamic evolution of prompts** – Prompts, which are central to generative AI application behavior, are not static elements. They are critical software artifacts that undergo continuous evolution. Their evolution demands rigorous lifecycle management, versioning, and proper decoupling from the application code itself.

- **Bridging the prototype-to-production gap** – PoCs are often developed under ideal conditions with clean data and manual processes. This can cause developers to overlook the challenges of production applications, such as scalable infrastructure, continuous monitoring, enterprise-grade security, and real-world data variability.
- **Lack of observability** – Without comprehensive tracing and observability mechanisms, it becomes difficult to determine why generative AI applications fail. This lack of deep insight hinders effective troubleshooting and continuous improvement.
- **Missing evaluation frameworks** – Unlike PoCs, which may rely on subjective assessments, production systems necessitate automated, objective evaluation frameworks. These are crucial for continuously measuring quality, detecting performance regressions, and validating that the application consistently meets its objectives.
- **Absent governance and guardrails** – PoCs typically do not incorporate the essential compliance controls, security guardrails, and data privacy measures that are non-negotiable for enterprise-level deployment. This oversight can lead to significant risks in production environments.
- **No clear path to return on investment (ROI)** – Without a well-defined framework for measuring costs (such as token usage and infrastructure expenses) and demonstrating business impact, generative AI projects often struggle to prove their value and secure sustained funding and support.
- **New threat landscape** – Generative AI applications introduce novel and sophisticated attack vectors, including prompt injection, jailbreaking, and data poisoning. These require specialized security strategies and proactive mitigation efforts beyond traditional cybersecurity measures.
- **Rapidly evolving technical stacks** – The generative AI field is characterized by constant innovation, new tools, and rapid technical stack advancements. Traditional development frameworks often struggle to keep pace with these changes.

These challenges are deeply interconnected, making it ineffective to address them in isolation. The non-deterministic nature of LLMs requires robust evaluation and observability, and successful production deployment demands integrated governance, security, and scalability from the start. Organizations need a flexible framework that can quickly adapt to new developments and architectures, accommodate shorter development lifecycles, and integrate emerging capabilities while maintaining operational stability. GLOE addresses these interconnected challenges through a comprehensive, integrated framework that manages the entire lifecycle, offering a unique value proposition for organizations that are implementing generative AI solutions.

Guiding principles of the GLOE framework

GLOE is founded upon a set of core principles that are designed to promote the development and operation of robust, scalable, and responsible generative AI applications:

- **Iterative and evidence-backed development** – GLOE promotes data-driven decision making through continuous experimentation and evaluation. Each development cycle builds upon validated learnings. This systematic approach makes sure that investments are backed by empirical evidence rather than assumptions.
- **Continuous improvement and feedback loops** – GLOE operates on the fundamental principle that operationalizing a generative AI application is not a one-time event but an ongoing process of continuous improvement. This process is actively fueled by real-world usage and a robust feedback system. This includes the implementation of automated feedback loops, direct user feedback mechanisms, and strategic human-in-the-loop interventions.
- **Holistic quality assurance** – To help address the inherent non-deterministic nature of generative AI outputs, GLOE mandates a multi-faceted evaluation strategy. This approach combines automated metric-based assessments, model-based evaluations (such as LLM-as-a-judge), and human evaluation to promote a comprehensive quality assessment.
- **Security and governance by design** – In the GLOE framework, security, regulatory compliance, and responsible AI principles are deeply integrated throughout the entire application lifecycle, rather than being treated as afterthoughts. This encompasses multi-layered guardrails, proactive adversarial testing (red teaming), and robust, auditable trails.
- **Modular and cloud-native architecture** – GLOE advocates for an architectural approach that involves decomposing complex generative AI applications into reusable microservices. This aligns with cloud-native principles, and it fosters enhanced scalability, resilience, and cost-effectiveness in deployed systems.
- **Automation with human oversight** – GLOE promotes extensive automation through continuous integration and continuous delivery (CI/CD) pipelines for all generative AI artifacts. It also emphasizes the critical and irreplaceable role of human-in-the-loop interventions.

These guiding principles collectively reflect a mature engineering discipline applied to the field of generative AI. By systematically applying these principles, GLOE elevates generative AI development from ad-hoc scripting and experimental phases into a rigorous engineering practice. This disciplined approach enables more predictable outcomes and facilitates broader enterprise adoption.

Personas who benefit from GLOE

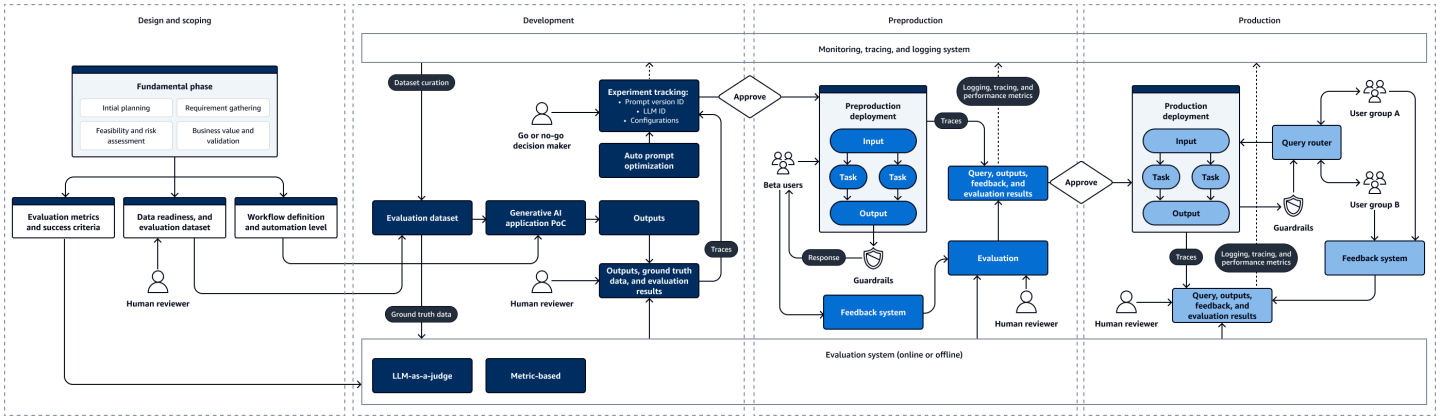
GLOE is designed to serve various roles and stakeholders who are actively involved across the generative AI application lifecycle. The framework provides tailored guidance and solutions for each persona to help foster a collaborative environment. The following are these personas:

- **Generative AI application developers** – These individuals are responsible for building the core generative AI application logic, performing prompt engineering, and integrating with various LLMs. For this audience, GLOE provides structured workflows for effective prompt management, comprehensive evaluation, and streamlined deployment processes.
- **Data scientists and AI engineers** – This group focuses on model selection, rigorous evaluation, and (if necessary) model fine-tuning. For this audience, GLOE offers robust mechanisms for experimentation tracking, comprehensive evaluation frameworks, and processes for validating data readiness.
- **Platform engineers and DevOps engineers** – Tasked with constructing and maintaining the foundational infrastructure and deployment pipelines, these engineers benefit from GLOE's guidance on modular architectures, generative AI-specific CI/CD practices, deep observability, and scalable maintenance.
- **Information security engineers (ISEs)** – Their primary concern is validating the security and compliance of generative AI systems. GLOE provides this persona with frameworks for implementing multi-layered guardrails, conducting adversarial testing (red teaming), and establishing auditability.
- **Business leaders and product managers** – These stakeholders are focused on demonstrating tangible business value, effectively managing risks, and achieving a clear ROI. For this persona, GLOE offers structured proof of concept (PoC) validation processes, detailed cost modelling, and clear go/no-go decision criteria to support their strategic objectives.

The comprehensive coverage of these diverse personas within the GLOE framework inherently fosters cross-functional collaboration. The challenges identified previously (such as a lack of observability, absent governance, or an unclear ROI) frequently arise from communication breakdowns or misalignments between different functional teams. By explicitly addressing the unique needs and concerns of each persona within a unified framework, GLOE promotes a shared understanding and a more collaborative workflow. This collaborative environment is crucial for successfully moving generative AI projects from PoC to production.

About the GLOE stages

The GLOE framework provides a holistic and structured approach to managing the generative AI application development lifecycle. As shown in the following diagram, it operates as an iterative cycle that emphasizes continuous refinement based on data and feedback gathered from subsequent stages. This framework organizes the generative AI application lifecycle into three distinct yet interconnected stages: development (PoC and experimentation), preproduction (validation and staging), and production (deployment and continuous operations). Each stage is characterized by specific objectives, key activities, and operational considerations. All are designed to help you transition from an initial concept to a live, high-quality application., high-quality application. The following diagram shows the stages and key components of the GLOE framework.



Development stage

The development stage is also known as the *PoC and experimentation* stage. The purpose of this stage is to validate the core concept, establish foundational elements, and iteratively refine your prompts or model. The key activities in this stage include:

- Manually or automatically optimizing prompts
- Tracking and versioning experiments
- Defining an evaluation dataset
- Performing human labeling of a dataset
- Evaluating through techniques such as LLM-as-a-judge
- Creating CI/CD pipelines for development

Preproduction

The preproduction stage is also known as the *validation and staging* stage. The purpose of this stage is to validate the application with internal or beta users, prepare for production deployment, and establish feedback systems. The key activities in this stage include:

- Designing modular resources
- Establishing a feedback system
- Performing online validation
- Performing unit, integration, and end-to-end testing
- Establishing an AI gateway
- Setting up tracing and logging systems
- Defining a deployment strategy
- Establishing guardrails and conducting adversarial testing

Production

The production stage is also known as the *deployment and continuous operations* stage. The purpose of this stage is to deliver value to the end users of the application. In this stage, you promote continuous performance, manage updates and changes, and maintain operational excellence. The key activities in this stage include:

- Collecting feedback
- Monitoring continuously
- Performing shadow testing
- Defining and following rollback strategies
- Deploying iteratively
- Managing governance and security risks
- Tagging for cost allocation
- Validating flexible scaling
- Implementing CI/CD practices

GLOE stage 1: Development, PoCs, and experimentation for generative AI applications

Generative AI is opening new frontiers of innovation across industries, and the development stage is where these possibilities begin to take concrete shape. The Generative AI Lifecycle Operational Excellence (GLOE) framework offers a structured approach to this exciting phase. It guides teams from initial concept to a functional proof-of-concept (PoC). The development stage, also called the *proof-of-concept (PoC) and experimentation* stage, is characterized by exploration and iterative learning. It helps organizations tap into the potential of generative AI in a controlled, low-risk environment.

This stage covers the key components that make up a successful generative AI PoC. From strategic planning and prompt engineering to evaluation frameworks and technical implementation, this section provides a comprehensive roadmap for embarking on a generative AI journey. Whether you're looking to enhance customer experiences, streamline operations, or create entirely new products and services, the development stage sets the foundation for turning AI-powered ideas into reality.

Building on these foundations, this chapter is organized into three main sections:

- Architecting a successful generative AI proof of concept – This section focuses on architecting a successful PoC, covering essential elements like business value validation, data readiness assessment, and technical feasibility studies.
- Developing and experimenting with generative AI – This section delves into the hands-on aspects of development and experimentation, providing detailed insights into model selection, prompt engineering, context optimization, and systematic evaluation strategies.
- Security controls for the development stage – This section describes foundational security controls for generative AI applications in the PoC stage. It discusses threat modeling, access controls, and supply chain security.

Throughout these sections, you'll find practical frameworks and proven methodologies that have helped teams successfully navigate their generative AI initiatives. When you have finished architecting and developing your PoC, use the recommendations in the Advancing a generative AI PoC to preproduction section to determine whether to progress to the next stage.

Architecting a successful generative AI proof of concept

The journey to a production-grade generative AI application begins long before the first line of code is written. It starts with a strategically sound proof of concept (PoC). Too often, PoCs are treated as technical demos that are designed to impress rather than as rigorous experiments that are designed to promote learning. A successful PoC is a strategic tool that derisks investment by validating the following: business value, data readiness, technical feasibility, and risk mitigation. This framework transforms the PoC from a technical exercise into a comprehensive business validation process. This helps you make sure that any "go" decision is backed by evidence, not just enthusiasm.

This section contains the following topics:

Demonstrating business value

In this phase, you connect business goals with technical metrics. A PoC must first and foremost answer the question: "If we build this, will it matter?" This requires translating a high-level business objective to specific, measurable success criteria. The following actions can help you validate business value:

1. **Start with strategic business objectives** – A generative AI initiative must be anchored in the organization's overarching goals. Primary drivers might be to increase operational efficiency, enhance customer experience, accelerate innovation, or create new revenue streams. Top-down alignment with business objectives helps you make sure that the project is strategically relevant.
2. **Translate value with the OGSM framework** – Tracking technical metrics that are disconnected from tangible business impact is a common failure point in generative AI projects. A robust evaluation strategy requires a clear, structured framework that links high-level business goals to specific, actionable metrics that engineers can use to optimize the system. The Objectives, Goals, Strategies, and Measures (OGSM) framework can provide this structure. It helps you make sure that every technical measurement is traceable back to a meaningful business outcome. It consists of the following levels

a. **Objectives (the *why*)** – Objectives are the overarching business intent or strategic intent. An example is improving customer support efficiency.

b. **Goals (the *what*)** – Goals are the quantifiable business targets that track progress toward the objective. An example is reducing the average handle time by 20%.

c. **Strategies (the *how*)** – Strategies are broad approaches or initiatives to achieve the goals. An example is implementing AI-driven email summarization to accelerate support workflows.

d. **Measures (the *how we track*)** – Measures are the specific metrics, at both the system level (user experience) and the model level (developer-facing), that track performance against the goals. Examples include: first-contact resolution rates, user satisfaction scores, a hallucination rate, answer relevancy, or latency.

The following table provides a practical template for applying the OGSM framework. The examples in the template demonstrate how to connect objectives to metrics across different use cases. This translation exercise is a critical communication tool because it creates a shared language between business and technical teams.

Objective	Goal	Strategies	Measures
Improve customer support efficiency	Reduce average handle time by 30%	• Task completion rate • First-contact resolution • User satisfaction	• **Answer relevancy** – Is the answer pertinent to the query? • **Hallucination rate** – Is the answer grounded in the provided context? • **Latency** – How quickly does the user get a response?

Increase sales team productivity	Increase number of qualified leads generated per week by 15%	• Lead quality score, as rated by Sales team • Time to generate outreach draft • User adoption rate	• **Coherence or fluency** – Is the generated email well written and persuasive? • **Context relevance** – For Retrieval Augmented Generation (RAG), was the correct customer informati on retrieved and used? • **Toxicity or safety** – Is the tone appropriate and professional?
Reduce content creation costs	Decrease cost per marketing article by 40%	• First draft usability rate (percentage of drafts used with only minor edits) • Subject matter expert (SME) quality rating	• **ROUGE or BERTScore** – How similar is the generated text to a set of human-wri tten samples? • **Token usage** – What is the API cost per draft? • **Diversity** – Is the model producing varied, non-repet itive content?

For more information about validating business value, see following resources:

- KPIs for Generative AI: Measuring Business Impact & Strategic Value (Debut Infotech blog post)
- OGSM framework (OGSM)
- AI Proof of Concept (PoC): What It Is & How to Build One? (Quinnox blog post)

Validating data readiness

Generative AI outcomes are only as good as the data they are built on. A PoC must rigorously assess whether the required data is available, accessible, and of sufficient quality to support the use case. The following checklist can help you validate data readiness for a generative AI PoC:

- **Data privacy and security** – These are the highest priority. The default approach should be to use synthetic or fully anonymized data. If real data containing any potentially sensitive or personally identifiable information (PII) is necessary for the PoC, explicit approval from your information security and legal teams must be obtained before ingesting the data into any system. Ignoring this step is one of the most common pitfalls. For more information and privacy and the how to anonymize data on AWS, see the Personal Data OU – PD Application account in the *AWS Privacy Reference Architecture*.
- **Data inventory and access** – Catalog all data sources that are relevant to your PoC and assess their accessibility, reliability, and integration complexity. Document data lineage to understand the origin and transformation history of your datasets. Establish clear data lineage tracking to enable auditing and troubleshooting throughout the PoC lifecycle.
- **Ground truth data curation** – For generative AI PoC projects, preparation of an evaluation dataset replaces traditional curation of training data curation. You need high-quality test datasets so that you can assess AI outputs against validated benchmarks and ground truth references.
- **Reference answer validation** – Ground truth must achieve high factual accuracy through SME validation. Validation should incorporate historical workflow data, expert-annotated examples, and verified documentation.
- **Business context alignment** – Datasets must comprehensively represent target business scenarios with contextual richness that aligns to PoC objectives. Customer service applications require conversational context and multi-turn dialogue patterns, while document processing needs structured content hierarchies and domain-specific terminology.

- **Data currency and relevance** – Establish data freshness requirements based on application sensitivity. Verify temporal relevance to prevent outdated information from generating misleading AI content. Document refresh cycles and establish version control.
- **Multi-modal completeness** – For text, image, or code generation applications, ground truth must encompass all relevant output modalities with cross-modal consistency validation. Define format-specific quality standards and confirm comprehensive coverage across different content types to identify model limitations and enable accurate cross-format evaluation.

For more information about validating data readiness, see the following resources:

- How to build a successful AI PoC: A step-by-step guide (Azilen blog post)
- A comprehensive guide to generative AI implementation for enterprises (XTM blog post)
- Data quality for successful generative AI program implementation (Wipro article)
- Data security, lifecycle, and strategy for generative AI applications (AWS Prescriptive Guidance)

Assessing technical feasibility

After you have defined the business value and confirmed data readiness, the PoC must prove that the solution is technically buildable within the organization's specific environment and that it can meet the goals outlined in the initial business case. This includes evaluating the integration landscape, making pragmatic technology choices, and confirming that the team has the right skills.

The PoC should test technical feasibility against these goals, and at its completion, the learnings and metrics across different technical dimensions should inform updates to the business case and potentially refine the functional requirements.

The following are the key dimensions of technical feasibility:

- **Focus on rapid iteration and model-task fit** – Start with a solid, general-purpose base model and prioritize prompt or context engineering over chasing the perfect model. Choose a few models, and then test them for suitability. Consider modality, size, cost, context window, tool use, output speed, and multilingual support. For feasibility, consider starting with higher-cost, higher-performance models, then evaluate lower-cost options to balance cost, accuracy, and latency.
- **Evaluate integration** – Assess the integration landscape and non-functional requirements (NFRs), such as privacy, security, scalability, and maintainability. Include performance metrics, such as end-to-end latency, requests or tokens per second, and concurrency to align system

design with application needs, such as real-time or batch operations. For Retrieval Augmented Generation (RAG), evaluate managed and self-hosted vector database options for latency and scalability. For agentic AI use cases, validate the reliability, latency, and security of external APIs or tools that the agent will use.

- **API compared to self-hosted** – Choose between commercial, self-hosted, or open source APIs based on NFRs, such as privacy, compliance, cost, and customization. Factor in cost-per-unit economics, such as the costs for one thousand tokens, GPU hours, storage, and egress. Consider how they scale after deployment. Validate compliance with data regulations, content policies, and intellectual property rights.
- **Team capabilities** – Confirm the team has the technical skills to deliver the vision, including model evaluation, optimization, integration, performance tuning, safety measures, and adversarial input handling.

Choosing an AI approach

Before development begins, it's essential to select the primary generative AI approach for the PoC. Options include prompt engineering, RAG, agentic AI, and fine-tuning. This decision shapes the architecture, data requirements, and complexity of the project.

We recommend that you choose an approach as follows:

1. **Start with prompt engineering** – This approach is ideal for tasks that rely on the model's general knowledge and reasoning abilities without needing external, real-time, or proprietary information. Examples include summarization, content generation, and simple classification.

   For PoCs that use this approach, focus on rapidly iterating on prompt design and context engineering. This is the simplest and fastest starting point for most PoCs.
2. **Add Retrieval-Augmented Generation (RAG)** – Choose this approach if the application must provide factually grounded answers that are based on specific, proprietary, or up-to-date documents, such as internal knowledge bases or product manuals. RAG can help mitigate hallucinations by providing the model with relevant context. For more information about RAG, see What is RAG (Retrieval-Augmented Generation) and Retrieval-Augmented Generation for Large Language Models: A Survey (Arxiv).

   For PoCs that use this approach, in addition to prompting, the PoC must validate the retrieval pipeline. You must select an embedding model, choose a vector store, and optimize document chunking and retrieval strategies.

3. **Evolve to agentic AI** – Consider agentic AI for complex, multi-step tasks that require the model to do more than just answer a question. Agents can interact with external tools, APIs, and data sources to accomplish a goal. For example, they can diagnose a technical issue by running commands. For more information, see Agentic AI (AWS Prescriptive Guidance).

   For PoCs that use this approach, define the agent's goals and the available tools. The PoC must test the agent's ability to reason, plan, and reliably use its tools. This involves selecting an agentic framework, such as Strands Agents, and managing the security and reliability of the tool integrations.
4. **Consider fine-tuning** – Consider this approach if you need to teach the model a specific style, format, or niche terminology that is difficult to replicate through prompting or RAG alone. It can also be used to improve performance on a very narrow, repetitive task. Before you choose this approach, evaluate the significant costs (time and money) of the fine-tuning process compared to the expected performance improvements. For most knowledge-injection use cases, RAG is a better starting point than fine-tuning.

   For PoCs that use this approach, the PoC must focus on curating a high-quality training dataset of prompt-completion pairs.

Your choice of approaches does not have to be mutually exclusive. A sophisticated system might combine all these techniques. However, for a PoC, it's crucial to start with the simplest method that can validate the core business value.

Mitigating project delivery risks

Generative AI PoC projects face unique risks that differ from traditional software development projects. Unlike deterministic systems, generative AI applications produce variable outputs that can be difficult to predict or control, making early risk identification and mitigation critical for project success. Additionally, the experimental nature of generative AI technology means that teams often work with rapidly evolving tools, uncertain performance benchmarks, and unclear cost structures.

This section outlines a pragmatic governance approach specifically designed for the PoC phase. It focuses on the most critical risk factors that can derail generative AI projects before they reach production. Rather than implementing heavy-weight processes that slow innovation, it emphasizes lightweight but essential practices that help teams:

- Catch quality and performance issues early before significant resources are invested
- Maintain stakeholder alignment through structured feedback mechanisms

- Make data-driven decisions about whether to continue, pivot, or terminate the PoC
- Establish realistic expectations for costs, timelines, and technical feasibility
- Build confidence in the approach before scaling to production

The goal of governance during a PoC is to pragmatically address critical showstoppers early in the process. This lightweight approach focuses on a few non-negotiable areas:

- **Feedback loops** – From the beginning, you must establish a simple, structured process for collecting feedback from the business leads, domain SMEs, and other key stakeholders. This could be as simple as a shared spreadsheet for reviewing outputs, a dedicated Slack or Microsoft Teams channel, or a minimal UI with positive and negative feedback ratings. The key is to make the feedback process consistent, timely, and actionable for the engineering team.
- **Iterative development with clear outcomes** – Use short, outcome-focused iterations to manage risk and enable quick adjustments. This helps teams learn and adapt without committing excessive resources too early.
- **Exit criteria** – Set clear thresholds for quality, latency, and cost, If results fail to meet these, pivot or end the PoC so that poor performance does not lead to wasted effort or sunk costs.
- **Scoped testing** – Validate only the core components before you invest in full end-to-end integration. This reduces complexity and isolates potential failure points early.
- **Operational readiness** – Address data quality and retrieval issues before prompt tuning, and track unit economics early. *Unit economics* are the per-request costs, which includes token usage, compute resources, and storage. Unit economics determine your application's financial viability at scale.

Developing and experimenting with generative AI

After you have established a strategic foundation per the Architecting a successful generative AI proof of concept section, the project transitions to technical execution. The goal of the PoC is to validate the core AI value proposition as quickly and cheaply as possible.

At this stage, the platform should be minimal because you should prioritize speed of iteration over engineering purity**.** The architecture should be simple and use managed services to the greatest extent possible. For an example of a typical RAG PoC, this might be a single AWS Lambda function that orchestrates API calls to an LLM through Amazon Bedrock and a managed knowledge base, such as through Amazon Bedrock Knowledge Bases. The focus is entirely on the application logic—

the prompt, the retrieval strategy, the evaluation metrics, and the output parsing—not on building and managing infrastructure.

Development should occur in environments that facilitate rapid experimentation, such as simple Python scripts. There is no need for a formal CI/CD pipeline, extensive monitoring, or automated deployment at this stage. The objective is to enable a developer to change a prompt and re-run an evaluation in minutes. The lightweight, iterative development loop centers on rapidly improving the quality of the model's outputs through disciplined prompt and context engineering, which is guided by focused evaluation mechanisms.

The technical execution of a generative AI PoC follows a logical progression that mirrors the scientific method: hypothesis, experimentation, measurement, and iteration. This journey begins with selecting the right foundation model for initial experiments, progresses through increasingly sophisticated prompt and context engineering, and culminates in systematic optimization based on evaluation results. Each step builds upon the previous, creating a compound learning effect that transforms initial assumptions into validated solutions.

This section contains the following topics:

Choosing models for generative AI applications

Every generative AI PoC begins with a fundamental question: which model should I use? This isn't about finding the best model in absolute terms, but rather identifying the right starting point for your specific use case and constraints. The model you choose becomes the engine that powers all subsequent experimentation. This decision is foundational to development velocity and cost efficiency.

Start by mapping your use case requirements to model capabilities across multiple dimensions:

- **Functional requirements mapping** – Identify non-negotiable capabilities upfront. If your PoC requires analyzing images of documents, you need multi-modal support. If it involves

orchestrating multiple API calls, function-calling capability is essential. For specialized domains (such as medical, legal, or non-English markets), consider models that have been pretrained on relevant datasets to reduce the prompt engineering overhead. These requirements can immediately narrow the field of viable models.

- **Performance and cost trade-offs** – Balance model intelligence against operational costs from day one. Although starting with a capable model helps establish quality baselines, simultaneously test smaller variants to understand the quality degradation curve. A customer support bot might achieve 95% accuracy with a large model at $0.50 per conversation, but some business might prefer 90% accuracy with a smaller model at $0.05 per conversation.
- **Latency considerations** – Different use cases have vastly different response time tolerances. Real-time applications (such as autocomplete functions or voice interactions) require sub-second responses. Analytical tasks (such as report generation and research) can accommodate multi-second latencies or longer. Test response times early, including network overhead and any pre or post-processing, to make sure that your chosen model can meet user experience requirements. Sometimes a smaller, faster model provides better overall user experience than a more capable but slower alternative.

We recommend the following to help you avoid common pitfalls when selecting a model:

- **Don't over-optimize prematurely** – The PoC phase is about proving feasibility, not achieving perfect optimization.
- **Don't ignore specialized models** – If you're working in a specific domain, test domain-specific models early.
- **Don't assume bigger is better** – Sometimes a smaller, faster model with well-crafted prompts can outperform a larger model for specific tasks.

Understanding context engineering for generative AI

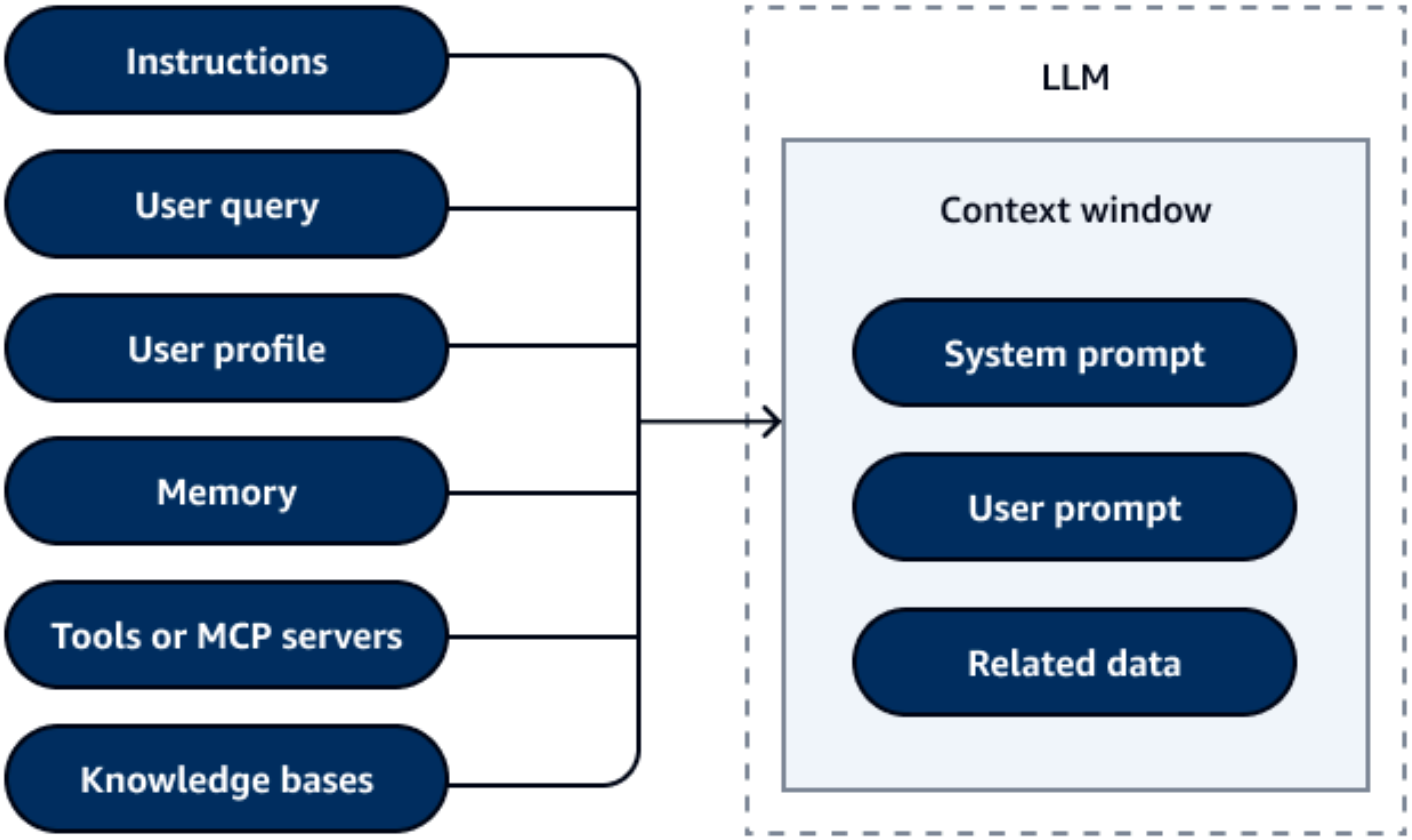
With the foundation model selected, the real experimentation begins. This is where abstract business requirements transform into concrete interactions with the models. The journey from basic prompts to sophisticated context engineering is rarely linear. It's an iterative discovery process where each experiment reveals new insights into what the model needs to perform effectively.

Most PoCs start in a simple, experimental environment, such as a Jupyter notebook or an Amazon Bedrock chat playground. The initial step is to translate the business problem into a direct

instruction for the LLM. This is basic prompt engineering. For instance, if the business problem is that a support team spends too much time summarizing long customer emails, the initial prompt might be a zero-shot instruction. For example, it might be "Summarize the following email into three bullet points." The team would test this prompt with a few real emails to see if the LLM understands the fundamental task. This process is iterative. Each cycle is informed by gap analysis, where you identify where the model falls short and then refine the prompt or evaluation criteria accordingly. For more information about prompt engineering, see Prompting best practices for Amazon Nova understanding models (Amazon Nova documentation) and Prompt engineering techniques and best practices: Learn by doing with Anthropic's Claude 3 on Amazon Bedrock (AWS blog post).

Teams often discover that simple instructions are insufficient for real-world business problems. The LLM might produce generic summaries, miss key domain-specific terms (such as product names or internal jargon), or lack crucial context from the customer's history. This is a critical learning moment in the PoC: the quality of the output is fundamentally limited by the quality of the input. This realization drives the need for a more sophisticated strategy. It necessitates moving from prompt engineering to the broader discipline of context engineering. *Context engineering* is the practice of designing systems that dynamically assemble the optimal set of information for an LLM to perform a given task. The decision to adopt a more complex context engineering strategy is a direct result of testing the initial, simpler prompts and finding them inadequate to solve the business problem to the required standard. For more information about context engineering, see Key components of a data-driven agentic AI application (AWS blog post) and Effective context engineering for AI agents (Anthropic).

Context engineering treats the context window not as a static input field, but as a dynamic workspace that is populated with precisely the right components. While prompt engineering focuses on what to ask, context engineering moves beyond the prompt text and focuses on what to show, presenting all of the relevant information that the LLM needs to succeed. During a PoC, the team experiments with which components to include in the solution to improve performance. The following diagram shows a high-level diagram of the components that you might include when context engineering.



A modern, well-engineered context payload is a composite of several dynamically assembled components, and each serves a specific purpose. The following describes the components and provides an example for the email summarization PoC:

- **System prompt or instructions** – This is the foundational layer that defines the LLM's role and the core task. An example might be the following instruction to a detailed persona: "You are an expert support agent that is summarizing customer emails for internal review." To encourage a consistent format, the team can use a one-shot or few-shot prompting technique, where you provide examples of the expected output format or structure, such as an ideal summary.
- **User query** – The actual input query. In this example, it would be the email from the customer.
- **User profile** – To generate a more relevant summary, the PoC might need to add information about the customer, such as their subscription level or history.
- **Memory** – Including background, such as the previous conversation history, in the context helps the LLM understand the ongoing conversation.
- **Tool definitions and Model Context Protocol (MCP) servers** – An advanced PoC might give the model a tool to look up information, such as the customer's account details that are stored in a customer relationship management (CRM) system.

- **Knowledge bases** – RAG is a powerful technique for providing additional context. If the email mentions a specific product, the system can retrieve technical documentation or FAQs about that product and add them to the context. This helps you make sure that the summary is factually accurate and grounded.

Generating a prompt template requires creating a reusable structure that combines these static and dynamic components. This transforms an ad-hoc process into a repeatable and testable part of the application.

Managing the generative AI prompt lifecycle

As central components of a generative AI system, prompts cannot be treated as simple text files that are unplanned and unmanaged. They are critical software artifacts that demand a rigorous, disciplined lifecycle management process, similar to application code. Establishing a formal lifecycle drives quality, reproducibility, collaboration, and safe deployment of prompt changes.

Use the following mechanisms to properly manage your prompts throughout their lifecycle:

- **Versioning** – Prompts must be versioned to track their evolution and enable rollbacks. Crucially, versioning a prompt goes beyond just tracking changes to the text. Each version's identity is defined by its performance. Therefore, a version number must be intrinsically linked to the comprehensive evaluation results it produced. This is described more in the following sections. For example, if the team discusses `customer-support-summarisation-v2.1.0`, they are referring to more than the template—they are also referring to its known set of performance metrics on a specific evaluation dataset.
- **Prompt store and registry** – To enable independent iteration and management, prompts must be decoupled from the application code. They should be stored in a centralized prompt store or registry. This can range from a version-controlled Git repository that contains YAML or JSON configuration files to a dedicated prompt management tool, such as Langfuse. This separation helps prompt engineers or even non-technical domain experts to update and experiment with prompts without requiring a full application redeployment.
- **Smart labeling and documentation** – Each prompt version requires clear and consistent metadata.
  - **Labeling conventions** – Adopt a structured format, such as `<feature>-<purpose>-<version>` to make the purpose of each prompt immediately obvious.
  - **Structured documentation** – For each version, maintain documentation that tracks its purpose, the rationale for changes, expected output format, and key performance metrics

from evaluations. This documentation is invaluable for debugging and onboarding new team members.

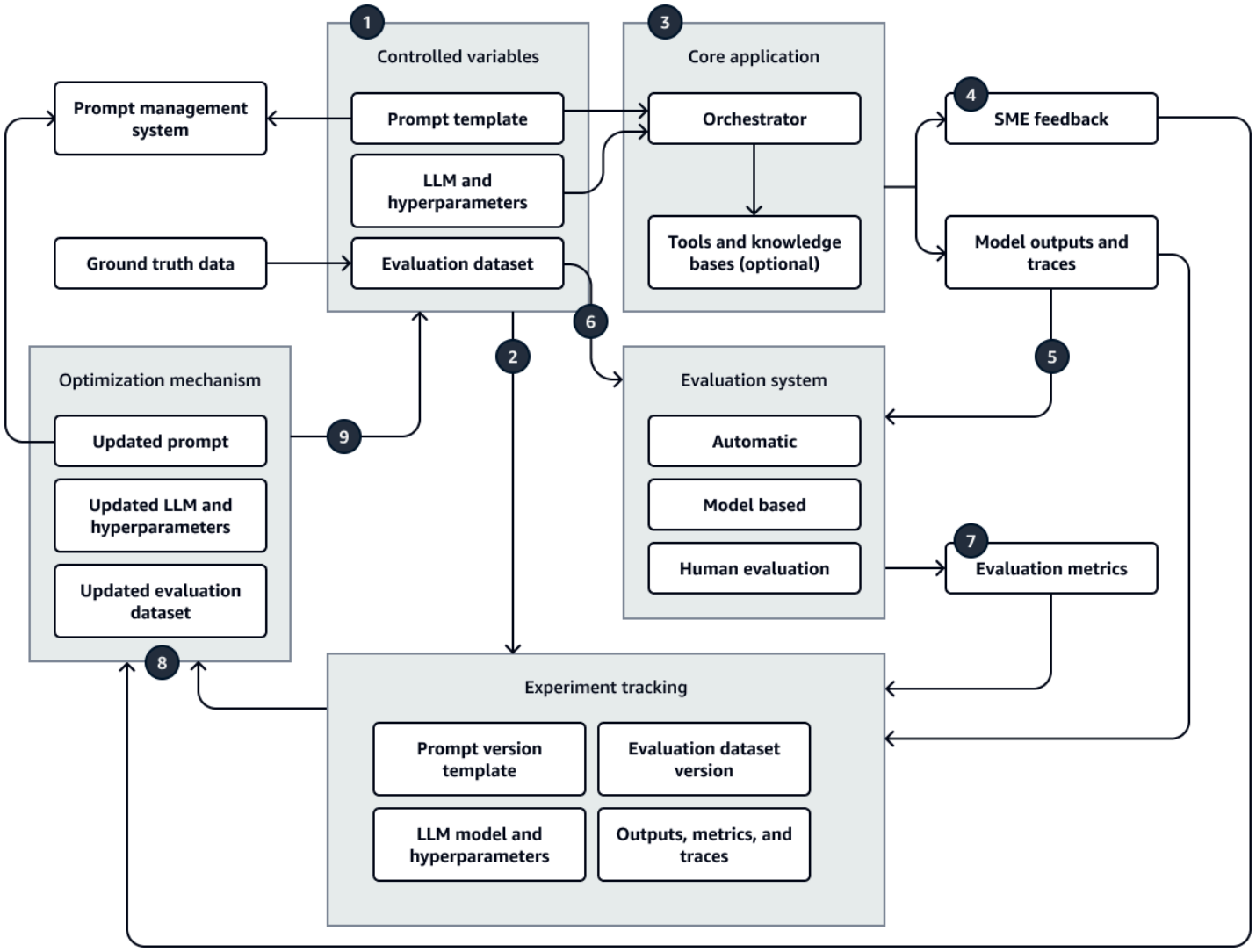
- **Collaborative workflows** – Prompt development should be a collaborative process. It's a best practice to implementing a pull request (PR)-style workflow for prompt changes, especially in staging and production environments. This allows for peer review, discussion, and quality assurance before a change is merged. It also creates an audit trail of who changed what, when, and why.
- **Environment promotion** – Just like application code, prompts must be managed across different environments. A typical promotion path is development to staging (or preproduction) to production:
  - **Development** – An environment for rapid iteration and experimentation.
  - **Staging** – A stable environment for rigorous testing and validation of prompt changes against a proven evaluation set before they are deployed to users.
  - **Production** – The live environment that serves users. Changes are promoted to production only after passing all tests in staging. This structured approach prevents experimental changes from accidentally impacting users and promotes a high-quality, reliable experience.

For more information about prompt management, see the following resources:

- Prompt versioning & management guide for building AI features (LaunchDarkly blog post)
- Prompt, agent, and model lifecycle management (AWS Prescriptive Guidance)

Creating evaluation loops for generative AI experimentation

The core of any successful generative AI PoC is a robust and repeatable development loop. This loop is the engine room where ideas are tested, prompts are refined, and quality is measured. While it shares conceptual roots with traditional machine learning operations (MLOps), the unique nature of generative models necessitates a new set of practices and tools. The following diagram shows an experimentation and evaluation loop for generative AI.



The diagram shows the following workflow:

1. Each iteration begins with controlled variables, which are the generative AI application inputs.
2. The controlled variables are stored for experimental tracking.
3. The core generative AI application accepts the inputs and produces outputs.
4. SMEs provide feedback on the outputs.
5. Outputs and traces that are captured from the generative AI application are stored for experiment tracking purposes, and they are consumed by the evaluation system.
6. The evaluation system consumes the evaluation dataset.
7. The evaluation system generates evaluation metrics and stores them for experiment tracking purposes.
8. The experiment results and SME feedback are used to update the application inputs.

9. The controlled variables are updated.

Traditional machine learning experimentation follows a relatively straightforward cycle. Data scientists modify hyperparameters, retrain models, evaluate performance on validation sets, and then select the best-performing variant based on classification metrics (accuracy or F1 score), regression metrics (mean absolute error or mean squared error) or clustering metrics. The process is deterministic - given the same input data and hyperparameters, traditional ML models generate identical results. This makes experiment comparison and reproduction straightforward. Version control focuses primarily on code changes, dataset versions, and model checkpoints, with experimentation costs concentrated in the training phase.

Generative AI experimentation operates fundamentally differently. It requires a more nuanced approach to tracking and versioning. The experimentation process centers around iterative prompt engineering, where small prompt changes can dramatically affect output quality and consistency. Unlike traditional ML, generative AI experiments must account for the non-deterministic nature of LLM responses. This requires multiple runs and statistical analysis to establish meaningful performance baselines. The experimentation lifecycle extends beyond model training to encompass ongoing inference optimization, where teams continuously refine prompts, adjust temperature settings, and experiment with different model variants to balance cost, latency, and quality. This creates a complex web of interdependent variables that traditional versioning systems weren't designed to handle. It necessitates specialized tracking approaches that capture the full context of each experimental iteration.

To move beyond ad-hoc experimentation, a structured and automated development loop is essential for building reliable generative AI applications. The previous loop diagram offers an example of a practical blueprint for this process. It establishes a systematic workflow where every change can be tested, measured, and reproduced. Each component of this loop plays an important role in this systematic approach to quality and iteration. The following are the primary components of this loop:

- **Controlled variables (inputs)** – Every scientific experiment begins with controlled variables. In generative AI, these are the versioned evaluation dataset, LLMs, hyperparameters, and the versioned prompt from the prompt management system. Treating the experiment variables as version-controlled inputs is beneficial for reproducibility. Changing any of the inputs creates a new experiment, and without strict versioning and tracking, it becomes difficult to compare results and determine if a change led to a genuine improvement.

- **Core application** – This is the generative AI application itself, configured with a specific prompt version and LLM model ID for the current run. This application orchestrates the logic, which could be a simple LLM call or a complex RAG system or agentic AI pipeline. The key is that its configuration is precisely defined and logged as part of the experiment parameters.
- **Experiment tracking** – This is the central hub where all experimental data is logged. It's the cornerstone of the entire loop. A single experiment is not just the output; it's a complete snapshot of the conditions that produce it. Tools like MLflow or Langfuse are designed for this purpose because they capture a holistic view of each experiment. Track the following variables:
  - **Prompt template version** – The unique ID of the prompt is logged as an input parameter. Or, you can integrate the prompt template version directly into the application code.
  - **LLM model and hyperparameters** – The exact model used and its inference parameters, such as temperature and top_p sampling, are logged. A change in any of these constitutes a new experiment. These parameters can also be tracked directly as part of the application code.
  - **Evaluation dataset version** – The version of the dataset used for the evaluation must be recorded to make sure that the test conditions are known.
  - **Outputs and metrics** – The raw text outputs from the model and the resulting performance scores from the evaluation system are logged as the experiment results.
  - **Execution traces** – For complex systems, such as RAG or agentic AI workflows, it's crucial to log the entire execution trace. This includes the retrieved documents, the final prompt sent to the LLM, and any tool calls made along the way. This provides deep observability for debugging and is a key differentiator from traditional ML tracking.

  For more information about experiment tracking, see Role of Experiment Tracking in Llmops (Lark).
- **Evaluation system** – This component takes the outputs from the application and compares them against the ground truth data, which is often curated by a human. This system is responsible for generating the quantitative and qualitative metrics that determine success. Given the nuanced nature of LLM outputs, this often includes model-based metrics, such as LLM-as-a-judge. However, it can also include automatic (calculation-based) or human metrics to assess aspects such as relevance or faithfulness at scale.
- **Optimization mechanism** – The final, most advanced stage of the loop involves feeding the evaluation results back into an automatic prompt-optimization component. This system can analyze failures and suggest or automatically generate a new, improved prompt version. You then check this prompt into the prompt management system to begin the next iteration. This creates a powerful, data-driven feedback mechanism that accelerates the path to a high-quality

application. Automatic prompt optimization is discussed in more detail in the Optimizing generative AI prompts section of this guide.

For more information about developing rapid evaluation and experimentat loops, see Generative AI app developer workflow (Databricks).

Evaluating quality and reliability in generated outputs

Evaluation is the most critical and challenging part of the generative AI development loop. Because the outputs are often unstructured and subjective, a multi-faceted evaluation strategy is required. The evaluation system itself should be treated as a product, a complex software component that needs to be designed, versioned, and validated to make sure that it provides a reliable signal of application quality.

This section contains the following topics:

Traits of a robust evaluation framework

It is essential to establish a high-quality, human-curated set of question-answer pairs early in the PoC phase of a new generative AI application. These gold-standard examples form the foundation of the testing framework and directly influence the accuracy and relevance of model evaluation throughout the development lifecycle. To build a strong evaluation framework, it is essential to start with a set of (ideally human-curated) question-answer pairs that serve as the gold standard for model behavior. The quality of these questions directly impacts the entire application's quality. It is critical that these gold standard outputs are correct. If you use generative AI to synthetically create the evaluation dataset, it's critical that you take extreme care to validate the results.

Successful generative AI frameworks usually share the following traits:

- **Use-case specificity** – The evaluation framework and the metrics chosen must be tailored to the specific business problem.
- **Rapid iteration** – The evaluation runs in seconds, enabling engineers to test hundreds of possible improvements daily. This rapid feedback loop accelerates innovation by removing bottlenecks at development speed.

- **Actionable feedback** – The generated score includes reasoned explanations, which provide engineers with precise insights into where the core application isn't functioning as expected. This transparency allows for targeted tuning of the evaluation system, uncovering hidden problems.
- **Configurable measurement** – The evaluation produces tunable numeric results, providing objective comparisons that are tailored to the specific workload. This adaptability helps you make sure that the numeric score accurately reflects what matters most for each use case, such as the relative importance of spelling accuracy (strict for legal documents, lenient for brainstorming), formatting consistency, or factual precision.
- **Comprehensive coverage** – The framework promotes reliable and consistent evaluation by using a high volume of diverse and categorized test cases. This broad coverage minimizes score variation and provides a trustworthy assessment across all possible scenarios, with results grouped by category for targeted feedback.
- **Component-level insight** – The evaluation framework assesses every step in the application, not just the result. This segmented approach provides granular visibility so that engineers can apply feedback to the exact portion of the workload where it will have the most impact. This requires a high-quality dataset for each individual step.

Evaluation datasets

The quality of an evaluation is directly tied to the quality of its dataset. A robust evaluation dataset should be diverse and representative of real-world usage.

The best practice is to build a high-quality, sometimes called *golden*, evaluation dataset from one or more of the following sources:

- **Real-world logs** – Extract examples from production or pilot user interactions, especially those that resulted in poor outcomes or received negative feedback. This makes sure that the evaluation focuses on real problems.
- **Manually curated examples** – Domain experts should craft a set of canonical examples that test the core functionality and critical business requirements.
- **Synthetically generated data** – Use an LLM to generate a wide variety of test cases, including edge cases, adversarial inputs (such as prompt injection attempts), and queries that are designed to test for specific biases.

The dataset must also cover a wide range of scenarios, including simple and complex queries, expected and unexpected usage patterns, and inputs with varying levels of ambiguity.

You can compare the outputs of the generative AI application to ground truth data. This helps you evaluate the quality and accuracy of the response. *Ground truth data* is data that is known to be accurate, verified, and representative of real-world outcomes.

For many generative tasks, defining a single, correct answer is impossible. This is a major departure from traditional machine learning, where labels are discrete. The following are two methods for handling ground truth data in generative AI evaluations:

- **Reference text** – Provide a complete, ideal response. This is useful when the exact wording or structure is critical, such as in regulated domains.
- **Expected facts** – A more flexible and robust approach is to list the key facts, entities, or concepts that must be present in a correct response. The evaluation system then checks for the presence of these facts, regardless of the specific phrasing. This accommodates the natural linguistic variability of LLMs.

Evaluation strategies and metrics

A generative AI evaluation strategy normally combines automated, model-based, and human evaluation approaches. The following table shows some metric options for each category.

Metric category	Metric name	Description	Typical use case	Requires ground truth data	Advantages and disadvantages
Metric-based	Recall-oriented understudy for gisting evaluation (ROUGE), such as ROUGE-L	Measures the n-gram overlap between the generated text and the reference text, focusing on recall.	Summarization and question answering	Yes, as reference text	**Advantages**: Fast, cheap, and objective **Disadvantages**: Misses semantic similarity and penalizes valid rephrasing

Metric-based	Bilingual evaluation understudy (BLEU)	Measures n-gram overlap, focusing on precision. Penalizes outputs that are too short.	Machine translation	Yes, as reference text	**Advantages**: Correlates well with human judgment for translation **Disadvantages**: Less suitable for creative or diverse text generation
Metric-based	Bidirectional encoder representations from transformers (BERT) score	Computes similarity between token embeddings of the generated text and the reference text.	General-purpose text generation	Yes, as reference text	**Advantages**: Captures semantic meaning better than n-gram overlap **Disadvantages**: Computationally more expensive

Model-based	Faithfulness or groundedness	Assesses whether the generated output is factually consistent with the provided source context.	RAG and fact-based question answering	No, uses source context	**Advantages**: Crucial for mitigating hallucinations **Disadvantages**: LLM-as-a-judge model can also hallucinate or be biased
Model-based	Relevance	Measures how well the response addresses the query or intent.	Chatbots and question answering	No	**Advantages**: Captures user-centric quality **Disadvantages**: Can be subjective, so a well-defined rubric is required in the LLM-as-a-judge prompt

Model-based	Coherence or fluency	Evaluates the linguistic quality of the text, assessing whether it is well-written, logical, and easy to read.	All text generation	No	**Advantages**: Measures a key aspect of the user experience **Disadvantages**: Highly dependent on the linguistic capabilities of the LLM-as-a-judge model
Model-based	Style or tone adherence	Checks if the output conforms to a specified style, such as whether it is professional, friendly, or in JSON format.	Brand voice and structured output	No	**Advantages**: Essential for brand consistency and system integration **Disadvantages**: Requires a very clear and specific LLM-as-a-judge prompt

Human	Preference score	A human evaluator rates the output on a scale (such as 1–5) or compares two outputs (such as A/B testing).	All use cases	Optional	**Advantages**: Especially advantageous for subjective criteria **Disadvantages**: Slow, expensive, and can be inconsistent

Model-based evaluation

The *LLM-as-a-judge* approach is a scalable method that uses an LLM to evaluate the outputs of generative AI system that uses a different LLM. It offers a cost-effective alternative to purely human evaluation, and it can accelerate development cycles. By reducing the need for human evaluation at every stage, which can be time-consuming, the automated process serves as a first filter to identify the most promising experiment results. This allows human reviewers to focus on the highest-value candidates, making reviews more efficient and speeding up decision-making.

However, it still requires careful prompt engineering and validation of the judge LLM against ground truth data, as it may inherit biases from the judge model. This makes it less reliable for highly subjective or safety-critical tasks. A human-in-the-loop approach is therefore recommended for critical decisions so that automated judgments are verified before deployment.

Just as human evaluation typically follows established criteria and converts subjective decisions into structured classifications, LLM-as-a-Judge transforms evaluation tasks into clear classification scenarios. Rather than seeking perfect score matches, the goal is achieving strong correlation with human judgment patterns.

A judge LLM receives evaluation prompts with specific criteria and assigns scores or classifications to system outputs. This works for single-output scoring (with or without reference) or pairwise comparisons. The key is framing evaluations as yes-or-no questions with explanations in order to validate alignment with human reasoning.

We recommend the following implementation best practices when using the LLM-as-a-judge approach:

- **Keep humans in the loop for critical decisions** – Ensure that outputs from automated evaluation are reviewed by human evaluators before being used for critical decisions or production deployment.
- **Validate against human patterns** – Test the performance of the judge LLM against human evaluation trends. Focus on correlation rather than exact matches.
- **Use a judge from a different model family** – To mitigate *self-preference bias*, where models tend to favor outputs that resemble their own, use a judge from a different provider or architecture. This is especially important when using this approach for model comparison. For more information, see Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena (Arxiv).
- **Define clear scenarios** – Precisely outline the evaluation context and what is being measured. Transform complex evaluations into structured yes-or-no decisions with required explanations. For example, use a rubric that defines specific key outcomes for each decision path, such as what qualifies a response as correct, partially correct, or incorrect. Describe how these outcomes determine whether the answer falls into the yes or no category.
- **Use categories** – Assess performance using defined categories (such as *excellent*, *good*, or *needs improvement*) rather than solely numeric scores. This can provide clearer insights and avoid overemphasis on minor score differences.
- **Prioritize models with strong reasoning capabilities** – To improve reliability and robustness, select judge models that have advanced reasoning skills. For more information, see A Survey on LLM-as-a-Judge (Arxiv).
- **Taking majority votes from repeated evaluations** – To mitigate bias and improve result reliability, run multiple evaluations and take the majority vote. For more information, see A Survey on LLM-as-a-Judge (Arxiv).
- **Use an ensemble of different models** – Combine the judgments of multiple models to enhance evaluation quality and reduce the risk of single-model bias. For more information, see A Survey on LLM-as-a-Judge (Arxiv).
- **Iterate and refine** – Continuously evaluate and refine the judge model's performance against human benchmarks.

Cloud platforms are beginning to integrate this functionality directly. For instance, Amazon Bedrock provides a list of built-in metric evaluator prompts for LLM-as-a-judge evaluation jobs.

Applications that go into production without a proper evaluation system or pipeline often suffer in the production stage. This can result in the need for model changes, such as replacing an LLM version that was deprecated by a vendor or addressing performance degradation. These situations highlight that the challenge is not just in building an evaluation system, but also in overcoming the temptation to skip evaluation or rely solely on human review. Such shortcuts create technical debt and hinder scalability in the future.

A critical best practice is to treat the evaluation prompts used by the judge LLM as important software artifacts. These prompts contain the rubrics and instructions for the judge model, and they should be version-controlled and managed in a prompt registry, just like the application's main prompts. The reliability of the entire automated evaluation system hinges on how well the judge's assessments align with human preferences. Therefore, it is essential to periodically validate the judge's performance against a high-quality dataset that is evaluated by human experts. Use correlation metrics to validate consistency.

After the PoC stage, teams can maintain a balance between rapid release cycles and reliability by incorporating LLM-as-a-judge into a stage-gate testing process. This could involve setting a threshold score in predefined categories that must be met or exceeded before progressing in the release pipeline. You can pair this approach with blue-green deployment strategies to minimize production risk. For example, this automated stage-gate approach could be used for minor or patch releases, which promotes speed and efficiency without overburdening the process. Major releases, on the other hand, might still require full or partial human evaluation as part of the deployment process to provide the necessary guardrails for safe and high-quality releases to customer-facing environments.

Human-based evaluation

Human reviewers are generally considered the ultimate source of truth. Automated and model-based evaluations are powerful for scaling, but they must be calibrated and validated against human judgment. Human feedback is essential for creating the initial high-quality evaluation datasets and for periodically auditing the performance of the LLM-as-a-judge to make sure that it aligns with human preferences and hasn't drifted. Although implementing an automated evaluation system is critical for efficient generative AI solution development, retain a human review as the final quality gate. A human should check the quality of the generated output before promoting the solution to the next stage.

Optimizing generative AI prompts

The evaluation results are not an endpoint; they are the fuel for optimization. By systematically analyzing the evaluation metrics and failure cases, teams can refine the prompts and context engineering strategies to enhance the quality, accuracy, and reliability of the LLM's outputs. This iterative refinement is the key to moving from a functional PoC to a high-performing application that meets the business requirements.

This section contains the following topics:

Prompt optimization techniques

Prompt optimization is the iterative process of refining prompts to improve the quality, accuracy, and reliability of LLM outputs. This process can range from manual, heuristic-based techniques to systematic, automated methods that treat prompt design as a formal optimization problem.

Baseline manual techniques

The following are foundational techniques that every practitioner should understand. They provide powerful levers for controlling model behavior with minimal complexity:

- **Zero-shot, one-shot, and few-shot prompting** – This spectrum of techniques relates to the number of examples provided in the prompt. With *zero-shot prompting*, the model is given an instruction without any examples, and it must rely on its pretrained knowledge. With *one-shot* or *few-shot prompting*, the prompt includes one or more examples of the desired input-output format. This is highly effective for guiding the model on specific tasks, tones, or structures. For more information, see Prompt engineering concepts (Amazon Bedrock documentation).
- **Chain-of-thought prompting** – *Chain-of-thought prompting* is a technique that breaks down a complex question into smaller, logical parts that mimic a train of thought. This helps the model solve problems in a series of intermediate steps rather than directly answering the question. This technique significantly can improve performance on tasks that require reasoning, such as math problems or multi-step logical deductions.

- **Task-specific prompting** – *Task-specific prompting* customizes instructions through frameworks that adapt to the task, domain, and output format.
- **Critique prompting** – *Critique prompting* uses iterative refinement methods, such as Self-Refine, Compressed Vocabulary Expansion (CoVe), and critic-guided. This technique helps models to generate, self-evaluate, verify, and improve responses in multiple steps.

For more information, see What is Prompt Engineering? (AWS) and Prompt engineering techniques and best practices: Learn by doing with Anthropic's Claude 3 on Amazon Bedrock (AWS blog post).

Automated prompt optimization

Automated prompt optimization (APO) represents the shift from prompt engineering as an art to a science. It employs algorithms to systematically search the vast space of possible prompts to find one that maximizes performance on a given task. This can minimize manual trial and error. The following are common APO techniques:

- **LLM-driven refinement** – This technique is also known as *metaprompting*. This powerful technique uses an LLM to optimize prompts for another LLM. A metaprompt instructs a capable model to act as a prompt engineering expert, critique a given prompt, and suggest improvements based on a set of failed examples. This creates an automated feedback loop where the AI helps refine its own instructions.
- **Heuristic and evolutionary search** – These approaches treat prompt optimization as a search problem. An initial prompt is considered a *gene*. The algorithm iteratively generates variations (*mutations*) and selects the best-performing ones based on evaluation metrics (*fitness*). It gradually evolves a highly optimized prompt. For more information, see A Survey of Automatic Prompt Optimization with Instruction-focused Heuristic-based Search Algorithm (Arxiv).
- **Programmatic frameworks** – Some frameworks, such as DSPy (GitHub) from Stanford University, are making optimization a core part of the development process. DSPy abstracts away the prompt itself. A developer defines the logic of their program as a series of modules, such as `signature('question -> answer')`. The DSPy compiler then automatically finds the optimal prompt (and can even fine-tune model weights) to make that program work effectively for a given dataset and metric. This approach makes prompt optimization more systematic, reproducible, and data-driven.

Self-optimizing and agentic systems

The manual and semi-automated optimization techniques used during the PoC are foundational, but as an application matures towards preproduction, the strategy for managing prompts must also evolve. The trend is a clear move away from manual crafting and towards greater abstraction, automation, and autonomy. This prepares the system for the complexities of a live environment. For example, Amazon Bedrock offers a prompt-optimization tool.

For prompt optimization, the following are trends toward self-optimizing and agentic systems:

- **Outcome engineering** – *Outcome engineering* shifts the focus from meticulously engineering the input (prompt engineering) to clearly defining the desired outcome. In the PoC, a team might manually craft a prompt to summarize a document. In a more advanced system, the goal becomes higher-level. An example might be "Generate a summary that achieves a 95% faithfulness score and is under 200 words." The system, using frameworks like DSPy, is then responsible for finding the optimal prompt to achieve that outcome.
- **Adaptive prompts for dynamic environments** – Production environments are not static. User needs change, new LLMs are launched, old LLM versions are deprecated, and new data becomes available. The next stage of development involves building systems with adaptive prompting capabilities, where the model can dynamically refine its own prompts based on real-time feedback and performance metrics. This is an improvement from the offline, batch optimization that is performed during the PoC.
- **The rise of agentic AI** – As systems become more complex, the single prompt-response paradigm gives way to agentic AI. An *AI agent* is an autonomous system that can plan, reason, and use tools to achieve a high-level goal. The PoC might have tested a single RAG call. The preproduction version might need an agent that can decide whether to call RAG, search the web, or query a database to best answer a user's query. This elevates the role of the engineer from a prompt crafter to an AI system architect, who defines the agent's goals, tools, and the ethical guardrails within which it must operate.

Thinking about these future-state capabilities during the PoC informs the architectural choices that you make early on. You must design the system to work now, but it also needs to effectively scale and evolve in the future. For more information, see The Future of Prompt Engineering: Evolution or Extinction? (Medium blog post).

Prompt optimization limits

Despite sophisticated prompt engineering and optimization techniques, evaluation metrics sometimes plateau below acceptable thresholds. This signals a fundamental mismatch between model capabilities and task requirements. This is a situation that no amount of prompt engineering can overcome. Recognizing when to pivot from prompt optimization to model reselection is crucial for maintaining development momentum.

The following are key indicators for model re-evaluation:

- Consistent failure patterns in specific capability areas, such as mathematical reasoning or long-context handling
- Quality metrics plateauing significantly below success criteria despite multiple optimization iterations
- Latency requirements are unachievable, regardless of prompt caching and decomposition

When these indicators emerge, shift from prompt optimization to targeted model selection. Use your accumulated evaluation data to identify specific capability gaps, then test alternative models or hosting options that directly address these weaknesses. This data-driven model selection, which is informed by extensive experimentation rather than initial assumptions, often unlocks performance improvements that prompt engineering alone cannot achieve.

Ensemble learning and voting

When high reliability is critical and a single model's output is inconsistent, consider using an ensemble approach as a more advanced optimization strategy. *Ensemble learning* is the practice of combining multiple LLMs to achieve performance that surpasses that of a single model. For more information, see Large Language Model Synergy for Ensemble Learning in Medical Question Answering: Design and Evaluation Study (JMIR Publications).

You can use an ensemble approach to perform model voting. *Model voting* is the practice of running the same input through multiple, diverse models. Then, you use a *voter* (or *aggregator*) mechanism to produce a final answer. This might involve:

- Having an LLM-as-a-judge pick the best response from the candidates.
- For fact-based tasks, extracting key facts from each response and taking the majority vote.

The model voting technique can help improve robustness and reduce the likelihood of a single model's idiosyncratic failure affecting the final output. However, it comes at the cost of increased latency and expense.

The iterative process between component optimization and model selection continues until the system meets the defined success criteria. This prepares the application for the transition from PoC to preproduction deployment.

Security controls for the development stage

The development stage requires foundational security controls that establish essential protections without impeding the rapid iteration and experimentation that characterizes the PoC objectives. At this stage, security focuses on building secure practices into the development workflow while maintaining the agility needed for innovation.

During system design, threat modelling provides the foundation for all subsequent security decisions. Conducting threat modelling early identifies security requirements before they become costly to implement. This helps teams understand the unique attack surface of generative AI applications across input processing, reasoning, and output generation. This analysis directly informs the implementation of basic guardrails. Adopting solutions such as Amazon Bedrock Guardrails with prompt injection filters can help provide immediate protection against common attack vectors while remaining lightweight enough for experimental environments.

Fundamental access controls can protect development environments, training data, and model artifacts through appropriate authentication mechanisms. They also help you securely store sensitive data and credentials. Development environments should implement role-based access controls that limit access to training datasets, model configurations, and experimental outputs based on team member responsibilities. Secure credential management and encrypted storage of sensitive artifacts can prevent unauthorized access while maintaining the collaborative nature essential for effective development work. Finally, development environments should be integrated with a AWS Security Hub CSPM to support organizational security observability needs.

Supply chain security assessment and basic data validation for training complete the development stage security posture. These controls address the integrity of third-party models, libraries, and data sources. They also implement validation processes to detect potential data poisoning or bias. The lightweight nature of these controls helps teams to establish security foundations early. This can help you avoid costly retrofitting while preparing for the more rigorous requirements of subsequent stages.

For more information, see the following resources:

- AWS Well-Architected Framework best practices:
  - Enforce encryption at rest
  - Identify threats and prioritize mitigations using a threat model
  - Grant least privilege access
  - Implement secure key management
  - Understand your data classification scheme
  - Use strong sign-in mechanisms
- Threat modeling your generative AI workload to evaluate security risk (AWS blog post)

Advancing a generative AI PoC to preproduction

The culmination of the PoC stage is a critical decision gate. It's the moment where the initial hypotheses are measured against real-world data. You make a formal, evidence-based decision about the future of the initiative. This is not a moment for intuition—it is a structured evaluation that determines whether to proceed with further investment, pivot the approach, or halt the project.

This section contains the following topics:

Evaluating the PoC outcome

You must present the final evaluation in the language of business value. Directly reference the objectives and measures that you established at the project's outset.

Consider the following when evaluating the success of the PoC:

- **Performance against key performance indicators (KPIs)** – The most important step is to measure the PoC's performance against the predefined success metrics. For example, did the

solution achieve the target time reduction, or did it meet the required accuracy or faithfulness scores? This quantitative analysis provides an objective measure of success.

- **Qualitative feedback and user experience** – What was the feedback from stakeholders and test users? Was the tool intuitive? Did it solve their problem in a way they found valuable? Qualitative insights often reveal critical nuances that metrics alone cannot capture.
- **Technical stability and scalability** – Did the PoC architecture perform reliably? While the PoC platform is minimal, this is the time to identify any fundamental technical roadblocks or scalability concerns that were discovered. This includes assessing latency, cost per interaction, and the reliability of data pipelines.
- **Key learnings and required changes** – No PoC is perfect. A core outcome is a list of key learnings. What assumptions were wrong? What new challenges were discovered? This analysis is crucial for accurately scoping the effort required for the next phase.

Making a go or no-go decision

With the evaluation complete, the team and stakeholders must make a formal go or no-go decision. This decision should be guided by a clear set of criteria that is revisited from the initial planning phase. To help you make this decision, answer the following questions:

- Has the PoC demonstrated that it can deliver meaningful business value and solve the intended problem?
- Is the technical solution viable, and is the path to a more robust implementation clear?
- Based on the PoC's performance and cost metrics, does the solution still have a positive ROI?
- Does the solution still align with the company's broader strategic goals?

If you confidently answer yes to these questions, then the project is ready to move from your development environment to a preproduction environment.

Preparing for preproduction

The transition from development to preproduction environments marks a significant step in maturity. To prepare for this transition, do the following to ready your PoC:

- **Formalize the architecture** – Evolve the lightweight PoC architecture into a more robust, modular, and secure design suitable for a pilot with real users.

- **Establish formal generative AI operations (GenAIOps)** – Implement a formal CI/CD pipeline for all AI artifacts, including prompts, configurations, and evaluation datasets. This helps you test and deploy changes in a controlled and repeatable manner.
- **Expand the evaluation suite** – Enhance your evaluation dataset with learnings from the PoC, including new edge cases and failure modes that you discovered during testing.
- **Prepare for a pilot** – Develop a simple but functional user interface and prepare to onboard a small group of pilot users to gather real-world feedback. This is the first step in the journey from a validated concept to a true minimum viable product.

GLOE stage 2: Preproduction, validation, and staging for generative AI applications

The preproduction stage is a pivotal phase in the GenAIOps lifecycle. It bridges the gap between a controlled experiment and a full-scale production deployment. This stage moves beyond the lab and into a controlled, near-production environment with a select group of internal or beta users. This allows teams to gather crucial feedback, harden the system, and build a final, data-driven business case for a full release.

The preproduction stage is defined by a fundamental shift in objective, from proving technical feasibility to validating business viability. While a PoC validates that the technology works, preproduction must answer far more critical business questions, such as "Is this solution valuable, reliable, secure, and scalable enough for enterprise use?" This requires moving beyond a simple technology experiment to a project with clear *production intent*, which is a commitment to building a solution that the business genuinely plans to deploy to solve a worthwhile problem. Success is now measured by the systematic mitigation of business, operational, and security risks.

This transition is where most generative AI projects stall, falling into the massive gap between a prototype and a production-ready system. PoCs create a dangerous illusion of completeness because they are often built under ideal conditions with clean data and manual processes. They ignore the work required for a real-world application. This includes building scalable infrastructure, establishing feedback loops for continuous improvement, integrating seamlessly with the user experience, implementing continuous monitoring, and fortifying the application with enterprise-grade security. For more information, see Is Your AI Project Ready for Production? A Practical Guide from PoC to Real-World Impact (Medium blog post).

Key failure points that emerge during this transition include but are not limited to the following:

- **Lack of observability** – The non-deterministic nature of LLMs means that they can excel in a demo and then break completely in real-world use. Without detailed tracing and observability, it is impossible to debug why it failed, not just what failed.
- **Missing evaluation frameworks** – PoCs are often judged subjectively. Production systems require automated, objective evaluation frameworks to continuously measure quality and detect regressions.

- **Absent governance and guardrails** – PoCs rarely include the necessary compliance controls, security guardrails, and data privacy measures that are non-negotiable for enterprise deployment.
- **No clear path to ROI** – Without a clear framework for measuring costs, such as token usage and infrastructure, and business impact, the project cannot demonstrate its value. This can lead to a withdrawal of funding and support.

Beyond these key failure points, there are additional ones, such as difficulty of use or challenges securing a high level of SME supervision or validation. The solution is to assemble a cross-functional production team from the early stage of the lifecycle. Involving stakeholders from IT, security, data science, and platform engineering during the PoC validation and at the start of preproduction is critical to prevent late-stage tech stack misalignment. It also helps you validate that the application is enterprise-ready. This early and continuous collaboration confirms alignment on critical technology choices, such as the foundational LLMs, vector databases, cloud platforms, and orchestration frameworks, before you invest significant development effort.

This chapter is organized into the following main sections to help you overcome the challenges of the preproduction stage:

- Architecting generative AI applications for production – This section focuses on architecting generative AI applications for production. It includes information about decomposing monoliths into microservices, designing data and prompt flows, implementing AI gateways, and optimizing for performance and cost.
- Hardening the generative AI application through a GenAIOps framework – This section describes how to harden generative AI applications through testing, security, and continuous improvement. It helps you implement CI/CD pipelines, observability, and feedback loops to prepare for production.

When you have finished architecting and hardening your application, use the recommendations in the Advancing your generative AI application to production section to determine whether to progress to the next stage.

Architecting generative AI applications for production

Architecting generative AI applications for production environments requires a sophisticated approach that goes beyond experimental implementations in a PoC. As organizations scale their

AI initiatives, they face complex challenges in creating robust, maintainable, and cost-effective systems. This section examines key architectural patterns and best practices that are essential for successfully deploying generative AI in real-world scenarios. With a focus on using a microservices architecture, implementing AI gateways, and optimizing cost and performance, the following sections provide practical strategies to build resilient generative AI systems. By adopting these principles, teams can create flexible, scalable infrastructures that are capable of meeting current demands while accommodating future growth and technological advancements.

This section contains the following topics:

Decomposing generative AI monoliths into modular and reusable microservices

Moving the generative AI PoC to a preproduction environment typically involves deploying its components into cloud-based resources. A common architectural anti-pattern in early generative AI development is the creation of a single, monolithic application that attempts to handle all aspects of a complex task in one application component. This approach is brittle and difficult to test. It's also extremely risky to update because a small change intended to improve one part of the output can have unforeseen and detrimental side effects on all other parts. A good practice is to move away from this monolithic approach and decompose the application's logic into a compound AI system, also known as a *chain*. For more information about compound AI systems, see The Shift from Models to Compound AI Systems (Berkley AI Research blog post). This architectural pattern involves breaking down a single large task into a sequence of smaller, discrete, and loosely coupled steps. Each step is handled by a more focused prompt or a dedicated tool.

The concept of a compound AI system aligns well with the principles of cloud-native architecture. *Cloud-native* refers to how a system is built and deployed. It emphasizes breaking down large applications into discrete, reusable, and independently deployable components that are known as *microservices*. Microservices are typically packaged in lightweight, portable containers. Adopting this approach for a generative AI application means that each component in the compound AI system (such as the RAG retriever, the summarization step, the data ingestion pipeline, and the user-facing frontend) can be developed, deployed, and scaled as a separate microservice. This

architecture enables superior resilience because the failure of one non-critical component does not necessarily bring down the entire system. It also enhances manageability and observability because each service can be monitored and updated independently.

In a modular generative AI architecture, each microservice is designed to perform a specific, reusable function within the overall system. This helps organizations to build a library of standard components that can be quickly assembled to create new and complex generative AI applications.

The following are common microservices that you can reuse for multiple generative AI applications:

- **Data ingestion and processing service** – This service is the data transformation hub for all knowledge. It is responsible for connecting to various data sources (such as databases or document repositories), cleaning the data, transforming it into a consistent format, strategically chunking it into smaller segments, and generating vector embeddings. By encapsulating this logic into a dedicated service, you create a reusable data pipeline that can feed multiple generative AI applications.
- **Model abstraction service (*AI gateway*)** – This service acts as a unified interface to various foundation models. It abstracts the specific API details of different providers and services, such as Amazon Bedrock or Amazon SageMaker AI. This decoupling is immensely powerful because it helps the organization switch between LLMs with a simple configuration change, without altering the core application logic. This service facilitates A/B testing of different models and supports granular controls. For more information, see the Centralizing control and observability with AI gateways section in this guide.
- **Orchestration service** – This service acts as the conductor, defining the workflow and the sequence of calls to the various task-specific agents and services. It manages the flow of data between components to fulfill a complex user request. You can use frameworks such as Strands Agents or LangChain to build this layer, or you can build a custom service for more control.
- **Memory as a service (MaaS)** – For sophisticated agentic systems, treat memory as a dedicated, decoupled service. A MaaS component provides a centralized and governable way to manage both short-term (*session*) and long-term memory. This approach breaks down the memory silos that form when each agent manages its own state. This allows memory to be shared and reused across different agents and interactions, which enables more coherent and personalized experiences. For more information, see What is Amazon Bedrock AgentCore? in the Amazon Bedrock documentation.
- **Gateway service** – This service serves as a central hub for securely connecting, deploying, and managing tools for AI applications and agents. It simplifies tool development by transforming enterprise resources into agent-ready tools. The gateway service provides unified access

through a single secure endpoint, features intelligent tool discovery with built-in semantic search capabilities, and offers comprehensive authentication for both inbound and outbound connections. For more information, see Amazon Bedrock AgentCore Gateway: Securely connect tools and other resources to your Gateway.

- **Feedback and logging service** – This service centralizes the collection of all user feedback (both explicit and implicit) and operational logs. By creating a single, reusable service for this function, you drive consistent data collection and monitoring across all generative AI applications in the enterprise.

The true power of a microservices architecture lies in decoupling and reducing the dependencies between components so they can operate and evolve independently. A key enabler of decoupling is the choice of communication protocols between services. The following are considerations when choosing protocols:

- Protocol independence
  - Asynchronous messaging, such as message queues and event streams, allows services to communicate without waiting for immediate responses. This reduces temporal coupling.
  - RESTful APIs provide standardized, stateless communication that doesn't require services to maintain connection state.
  - Event-driven architectures enable services to react to events without direct knowledge of event producers.
- Technology stack freedom
  - Well-defined communication protocols allow each microservice to be built using the most appropriate technology stack.
  - Services can be written in different programming languages if they adhere to the agreed communication contract.
  - Teams can choose databases, frameworks, and tools that best fit their specific service requirements.
- Evolutionary architecture
  - Proper protocol design allows services to evolve independently through versioning strategies.
  - New service versions can be deployed without breaking existing consumers.
  - Services can be replaced entirely if they maintain the communication contract.
- Scalability and resilience
  - Load balancing becomes possible when services communicate through well-defined protocols.

- You can implement circuit breaker patterns to handle service failures gracefully.
- Services can be scaled independently based on their specific performance requirements.

Managing asset promotion and environment transitions

One of the most critical aspects of generative AI development is the structured transition of validated artifacts between lifecycle stages. Unlike traditional software development, generative AI applications involve multiple interdependent components, such as prompts, model configurations, evaluation datasets, and application code. These components must be carefully coordinated and promoted together. Organizations that struggle with this transition can experience the following:

- Loss of experimental insights when moving from PoC to production environments
- Inconsistent configurations across different stages of development
- Quality regression when untested or unstable artifacts are promoted
- Broken feedback loops that prevent continuous improvement

The GLOE framework emphasizes continuous improvement and feedback loops and iterative and evidence-backed development. This asset promotion process is where these principles are operationalized. You must make sure that only validated, stable configurations advance and maintain channels for learning and refinement. The transition from the PoC stage to preproduction involves a structured governance process for promoting validated artifacts across environments:

- **Prompts and LLM versions** – Approved prompt versions, along with their associated LLM versions and hyperparameters (temperature, top_p, top_k), are promoted from the prompt store management solution in the PoC environment to the preproduction environment. This promotion follows version control principles that are similar to code deployment. The goal is to make sure that only stable, validated configurations advance to the next stage. The process emphasizes that prompt tuning and LLM selection should be primarily completed during the PoC stage, and preproduction should focus on infrastructure optimization and deployment tuning. Significant prompt or model alterations should occur in the PoC stage. This shift in focus prevents the common anti-pattern of continuing experimentation in environments designed for stability testing.
- **Application code deployment and integration** – The core application code orchestrates LLM calls, integrates with other services (such as vector databases for RAG), and handles user interactions. It should undergo structured deployment to the preproduction cloud environment.

This code represents the culmination of initial unit and integration testing conducted during the development phase, and it serves as the foundation for more comprehensive system-level testing in preproduction. The deployment process should make sure that all dependencies, service integrations, and orchestration logic are properly configured for the target environment. The application code also needs to maintain compatibility with the promoted prompt and model assets.

- **Evaluation datasets enhancement** – Evaluation datasets, including human-labeled ground truth data developed during the PoC stage, are systematically carried forward to preproduction environments. These datasets serve as the foundation for continued quality assessment. You can strategically extend them with real-world data that you collect from internal user interactions or controlled beta testing programs. This creates an enhanced evaluation dataset that provides more comprehensive coverage of actual usage patterns and edge cases. Importantly, this process includes bidirectional flow capability. The enhanced evaluation datasets can be returned to the PoC environment if performance requirements aren't met during preproduction testing. This supports iterative refinement without losing valuable insights.
- **Environment configuration and infrastructure setup** – Deployment configurations encompass the complete infrastructure specification required for preproduction operations. This includes cloud resource allocations, security configurations, and operational parameters. This also includes the secure management of API keys, environment variables, and service endpoints that enable proper integration with external systems and services. Infrastructure as code (IaC) templates support consistent environment provisioning and enable repeatable deployments while maintaining security and compliance requirements. The configuration management process establishes the foundation for scalable operations and provides groundwork for the production deployment.

Centralizing control and observability with AI gateways

As generative AI proliferates within an enterprise, leaving individual applications unmanaged can create a chaotic, unsecure, and expensive problem. Each team might use different models, manage API keys insecurely in their code, and have no visibility into costs or security risks. An *AI gateway* is a unified API gateway that streamlines access to multiple LLMs. The role of an AI gateway is to provide governance and security while improving agility, making it easier and faster for your teams to build AI applications.

The following are the key features and benefits of an AI gateway:

- **Unified authentication and security** – The gateway acts as a single, hardened entry point. It offloads the burden of authentication from individual applications, allowing the enforcement of a uniform, enterprise-wide security mechanism, such as OAuth. It securely manages all backend API keys and prevents them from being scattered across dozens of codebases.
- **Dynamic load balancing and routing** – A gateway can intelligently distribute requests across multiple model deployments. This can increase overall throughput by load balancing across different AWS Regions because requests per minute (RPM) and tokens per minute (TPM) limits are often Regional. It can also build resilience by routing traffic between different model providers. For example, it can fail over from Amazon Bedrock to self-hosted LLMs.
- **Centralized quota and cost management** – The gateway can manage complex rate limits (TPM and RPM) for different consumer groups. This prevents any single application from monopolizing resources. It becomes the central point for logging token consumption, which enables accurate cost tracking and departmental chargebacks.
- **Resilience and fallback strategies** – The gateway can implement sophisticated resilience patterns. A common strategy is called *spillover*, where traffic is primarily routed to a cost-effective, prepurchased provisioned throughput unit (PTU). If that PTU reaches its capacity, the gateway automatically redirects the overflow traffic to a more expensive but elastic pay-as-you-go endpoint. This promotes service continuity during demand spikes.
- **Centralized observability and logging** – The gateway is the ideal location to log all prompts, responses, latencies, and other performance metrics in a standardized format. You can feed this data directly into the central observability platform.

For more information about generative AI gateways and a solution to deploy an AI gateway on AWS, see Guidance for Multi-Provider Generative AI Gateway on AWS.

Considerations for AI gateways

When considering an AI gateway, to assess the needs for your AI eco-system, assess the following security, privacy, compliance, performance, reliability, and monitoring considerations.

Authentication

In general, the generative AI application access models through an API and authenticates through an API key. Your AI gateway should support authentication through the API key and securely store those keys. It should also provide a seamless way to integrate with your existing identity provider by using single sign-on (SSO) so that access is integrated with the larger ecosystem. Consider an AI gateway that also supports multiple protocols that are required by foundation models (stateless),

MCP servers , and other components of your application. Provide a means to consistently apply the authentication policies for such protocols.

MCP and data providers

Modern AI systems have more components than just an LLM. Data enrichment from existing systems makes the model provide more relevant results. As a result, organizations are exposing their data to the model in multiple ways, such as through RAG or MCP. The model might be hosted outside of your organization; therefore, adherence to the principle of least privilege becomes more critical. Your AI gateway should secure components and other data systems that the model accesses, and it must also secure the agentic applications and the associated access logs.

Fine-grained access control

Because the AI gateway is a central access point, it should separate authentication and tracking mechanisms based on association with groups or teams. This helps you manage multiple access policies that are suitable for different teams. Your gateway shall provide a way to manage different access policies to different set of users.

Guardrails

LLMs hallucinate, and you need control of the output they generate based on your business rules. Your gateway should provide a central point to capture outputs from LLMs and apply your guardrails in order to adhere to your organization's policies. Consider a gateway that allows data teams to write guardrails with ease, such as with a Python function, for easy integration and fast adoption of new business rules.

Auditing

For compliance and security, organizations might need to maintain a record of AI interactions. An AI gateway can act as a single point of entry for all LLMs and MCP traffic. It creates a log for every request, and it captures who made the request, which model was used, the full prompt and response, and any guardrails that were triggered. It can also track the cost.

Model choice

As the investment in foundation models grows, more models are becoming available for use in your applications. Open-source models or models from various providers might have their own protocol and formats to access the models. If your applications and teams are accessing multiple models, the time cost of switching between model protocols can become steep. Consider an AI gateway that provides the capability to translate from an open standard, such as OpenAI API, to

any other protocol. This can simplify applications and help your team test and experiment with new models.

Cost control and chargebacks

The financial governance of AI is fundamentally different from traditional APIs. Consider an AI gateway that provides cost controls based on AI-native metrics. Examples include token-based rate limiting and budget enforcement based on model price and token usage.

Dynamic routing

An AI gateway can serve as a control plane for model orchestration. It can address key challenges, such as resource availability and performance. Through intelligent routing strategies, the AI gateway can analyze incoming requests and route them to more appropriate models. There are different approaches to implementing a routing strategy. For example, you can use rule-based routing (such as time-based routing to use off-peak capacity) and more sophisticated semantic routing. Semantic routing might include *content-type intelligent routing* (where the AI gateway analyzes the content and intent of queries to determine the most appropriate model for each task) or *complexity-based smart routing* (where the gateway evaluates the complexity and sophistication required for each request before selecting the appropriate model).

Monitoring and alerting

A primary challenge in operating AI systems is the need for specialized monitoring to validate performance and reliability. Consider an AI gateway that captures LLM-specific interactions and sends alerts if things have gone wrong, such as slow responses. For seamless integration into an existing customer environment, the gateway must be able to export these metrics to the organization's established observability and incident management tools.

Designing generative AI applications for performance and cost

During the PoC stage, performance and cost are often secondary concerns. They are overshadowed by the drive to simply make the technology work. In preproduction, they become primary architectural drivers that influence every design decision. An application that is too slow or too expensive fails in production, regardless of how impressive its outputs are.

Latency reduction

Latency reduction in generative AI applications is crucial because users expect natural, real-time interactions. Additionally, lower latency often means more efficient resource usage, reducing costs for both providers and users. The following are latency reduction strategies:

- **Use streaming** – For applications with long-form text generation, waiting for the entire response can lead to a poor user experience. *Streaming*, where partial responses are sent to the UI as they are generated, dramatically improves the perceived speed of the application.
- **Smart caching** – Many user queries are repetitive. You can implement a caching layer in the AI gateway that stores and serves results for identical or semantically similar queries. This can significantly reduce both latency and cost.
- **Right-sized models** – A common mistake is to use the most powerful model, which is often the most expensive and slowest, for every task. A more sophisticated approach is to use a chain of models. A simple, fast, and cheap model can handle most requests. If it fails or lacks confidence in its answer, the request is escalated to a more powerful and expensive model. This tiered approach optimizes the trade-off between performance, cost, and quality.

Cost modelling

The preproduction stage must begin with the creation of a detailed cost model. This is not a rough estimate. It is a living document that is continuously updated and validated as the application is tested. The model should break down costs by the following categories:

- Estimated query volume and patterns, such as daily peaks
- For each type of query, the average token usage for both the prompt and completion
- The cost per token for the selected LLMs
- The cost of the underlying cloud infrastructure, such as compute for hosting, vector database storage and queries, and guardrails

Every architectural decision—from the choice of vector database to the selection of a foundation model—must be evaluated against this cost model. This practice forces the team to make conscious, data-driven trade-offs and provides the business with a predictable cost forecast for the production deployment.

Hardening the generative AI application through a GenAIOps framework

The evolution of generative AI applications from PoC to production-ready systems requires a comprehensive hardening process that promotes reliability, security, and continuous improvement.

Through the implementation of a GenAIOps framework, organizations can transform experimental prototypes into robust, enterprise-grade applications that meet stringent quality and compliance standards.

This transformation encompasses multiple key areas: formalizing the AI stack with rigorous versioning, implementing automated CI/CD pipelines, establishing deep observability, and creating multi-layered testing frameworks. These components work together to create a secure, governable system that can adapt and improve through carefully designed feedback loops.

This section contains the following topics:

Formalizing the generative AI stack

The approach to versioning generative AI assets undergoes a transformation between the PoC and preproduction stages. In the PoC, versioning supports rapid, wide-ranging experimentation to discover what works. In preproduction, the focus shifts from discovery to stability. Versioning becomes a formal, rigorous process for managing a now well-defined application stack. In this stage, you treat all generative AI artifacts with the same discipline as production code to promote reproducibility and auditability and to prevent quality regressions. For more information about versioning generative AI applications, see Version tracking for GenAI applications (MLflow).

The following are components that should be treated as versioned artifacts:

- **Prompts and model configurations** – During the PoC stage, the prompt registry acted as a sandbox for testing dozens of prompt variations and model parameters. In preproduction, the most successful prompt candidates and their associated model configurations are promoted and locked in as stable, versioned artifacts. They are no longer experimental strings, and you should treat them as configuration as code, which focuses on application settings and parameters (unlike infrastructure as code, which manages computing resources). At this stage, changes should be small, targeted tweaks that are intended to fix a specific edge case or improve a

performance metric. These changes must go through a formal review process, such as a pull request, and they must pass the full suite of automated evaluations in the CI/CD pipeline before being approved and promoted to production.

- **Application code** – The application logic, which evolved from a simple script in the PoC to a collection of microservices, is managed in a version control system, such as Git. Every deployment, evaluation run, and logged trace must be linked back to a specific Git commit hash. This provides an immutable record of the application logic, and this is the standard practice to which all other generative AI artifacts must now adhere.
- **Evaluation dataset** – This dataset is expanded with real-world examples, user-reported failures, and adversarial test cases that were discovered during early testing. It becomes the official benchmark against which all changes to the application stack are measured. Any modification to this dataset should be a versioned change. This makes sure that performance comparisons between different application versions are always fair and consistent.

This holistic approach to versioning means that a single application version is no longer just a Git commit. It is a complete, immutable snapshot of the entire stack: the specific code version, the prompt version, the model configuration, and the evaluation dataset version it was validated against. The application at this stage might also be open to internal user testing, which makes the online evaluation an important component in the lifecycle. For more information, see Establishing a multi-layered testing and evaluation framework in this guide.

Automating deployment through CI/CD pipelines

The manual deployment processes of the PoC stage must be replaced by a fully automated continuous integration and continuous deployment (CI/CD) pipeline. This pipeline is the engine that enforces quality, security, and consistency for every change so that only validated and hardened code reaches production environments.

A mature CI/CD pipeline for a generative AI application includes the following automated stages:

1. **Trigger** – The pipeline is automatically initiated by a new commit to the main code branch or by the promotion of a new prompt version in the prompt registry.
2. **Build** – The pipeline builds and packages the application's microservices into container images.
3. **Unit test** – The pipeline runs traditional unit tests against the deterministic components of the application, such as its data processing tools and API integration logic, to catch functional bugs early.

4. **Evaluation** – This is a critical, generative AI-specific stage. The pipeline automatically runs the new application version against the versioned evaluation dataset. It can use the same LLM-as-a-judge evaluation metrics that were developed during PoC stage in order to score the outputs. The pipeline is configured to fail the build if the evaluation scores drop below a predefined threshold. In this way, the pipeline automatically detects and prevents quality regressions.
5. **Security scan** – The pipeline runs automated adversarial tests (*red teaming*) against the application to check for common vulnerabilities, such as prompt injection attacks, personally identifiable information (PII) exposure, or system prompt extraction. For more information, see OWASP red teaming: A practical guide to getting started (Promptfoo).
6. **Deployment** – If all the preceding checks pass, the new version of the application is automatically deployed to a staging environment. This environment is a stable replica of production, used for final user acceptance testing, canary releases, or A/B testing.

Achieving deep observability

Observability is the practice of instrumenting a system to provide rich, detailed data about its internal state, allowing engineers to understand not just that a problem occurred, but why it occurred. The goals of observability evolve between the PoC and preproduction stages. The primary goal during PoC stage is debugging and validation. At the preproduction stage, the goal shifts to holistic system health and performance monitoring.

A robust observability strategy for generative AI requires the following three pillars of telemetry data, all collected and correlated in a centralized platform:

- **Unified metrics, logs, and traces** – In addition to collecting standard application logs and infrastructure metrics, you must also collect generative AI-specific data. This includes the full prompt and response payloads, token counts for cost tracking, tool call parameters and outputs, and user feedback signals.
- **End-to-end tracing** – For a compound AI system, a single user request can trigger a complex chain of events that involves multiple LLM calls, tool executions, and database queries. It's essential that you use end-to-end tracing tools, such as MLflow Tracing, Langfuse, or platforms that use OpenTelemetry. They capture the entire execution flow of a request, visualizing it as a *trace* that shows the inputs, outputs, and latency of each step. This is the single most powerful tool for debugging why an agent took an unexpected path or failed mid-execution.
- **Performance, cost, and quality monitoring** – The observability platform should provide real-time dashboards to monitor key business and operational metrics. These dashboards track not

only technical performance (such as latency and error rates) but also business-critical KPIs, such as the cost per request, token usage per user, and trends in quality evaluation scores and user feedback over time. This helps teams to proactively detect performance drift or escalating costs.

Establishing a multi-layered testing and evaluation framework

The preproduction stage is where the application's quality and robustness are put to the test. Given the non-deterministic nature of generative AI, a traditional testing strategy is insufficient. A multi-layered approach is required, one that combines foundational software testing with novel techniques that are designed specifically to validate the behavior of LLM-based systems. This rigorous testing is essential for building the confidence needed to deploy the application to real users.

The following table shows the core layers and components of a testing and evaluation framework. It describes *foundational* tests, which are traditional software testing practices that have been adapted for generative AI. It also describes *quality assessment* tests, which are specific to generative AI applications. These quality assessments form a specialized evaluation framework that can assess the quality of non-deterministic outputs. This hybrid approach combines automated offline analysis, live online testing, and continuous human feedback in the preproduction stage.

Layer	Test	Recommended practices
Foundational	Unit testing	• Use mock and stub dependencies • Aim for high coverage on the logic • Test for edge cases • Integrate with the CI/CD pipeline
Foundational	Integration testing	• Validate integration with the platform

		• Validate the data flow • Validate integration with external APIs
Foundational	End-to-end testing	• Focus on the workflow • Use realistic data • Automate the full flow from user interaction with the frontend to the entire backend system • Test variable response times from external APIs • Test asynchronicity from external APIs
Quality assessment	Offline evaluation	• Often uses LLM-as-a-judge approach • Integrate with the CI/CD pipeline
Quality assessment	Online evaluation	• Use A/B testing • Use canary releases

Quality assessment	Human-in-the-loop evaluation	• Collect explicit feedback • Collect implicit feedback • Close the loop

Unit testing

Unit testing, in a generative AI context, concentrates on the individual, deterministic tools or components within the larger system. Each function, such as a data-retrieval script, an API client, or a data-transformation function, must have its own suite of unit tests to validate its logic in isolation. For more information, see What is unit testing?

A primary challenge of unit testing is handling dependencies on external services, such as LLM APIs. Unit tests must remain fast and isolated, which means they cannot make live network calls.

The following are recommended practices for unit testing:

- **Mock and stub dependencies** – Use mocks and stubs to simulate the behavior of external services. For an LLM call, a mock can be configured to return a predefined, expected response. This helps you test how your code handles that specific output without calling the model.
- **High coverage on business logic** – Aim for high test coverage on the critical, deterministic code paths that prepare data for the LLM or process its output.
- **Edge case testing** – Actively test for extreme or invalid inputs (known as *edge cases*) to make sure that the component is resilient.
- **CI/CD integration** – Unit tests must be integrated into the CI/CD pipeline and run automatically for every code change in order to provide rapid feedback.

Integration testing

Integration testing is the layer that verifies the interactions between the modular components of the generative AI application and its underlying platform. While unit tests confirm that each component works on its own, integration tests make sure that they fit together correctly within a realistic environment.

A principle of preproduction integration testing is to consider the deployment platform itself. Testing on a local machine is insufficient because it cannot replicate the complexities of a cloud-native environment, such as network policies, IAM roles, and service-specific configurations. The best practice is to establish a dedicated preproduction or staging environment that mirrors the production technology stack as closely as possible. Use containerization and orchestration tools, such as Docker, to promote consistency.

The following are common enterprise integration scenarios that you should test:

- **Data pipeline integrity** – For RAG systems, this is a top priority. Tests must validate the entire data flow, from the ingestion service that pulls data from a source system, through the processing and embedding service, and into the vector store. This catches issues that unit tests would miss, such as data format mismatches or errors in the ETL process.
- **Agentic tool use and external APIs** – Tests must verify that an agent can correctly authenticate with and call external tools or enterprise APIs, parse their responses, and handle potential errors gracefully. For example, you might test integration with a customer relationship management (CRM) system or a booking system.
- **Authentication and authorization flows** – Verify that the application correctly integrates with enterprise identity providers and that role-based access controls are properly enforced across service-to-service communication.

The following are recommended practices for integration testing:

- **Isolate with service virtualization** – When a full production-mirror environment is too complex or expensive for every test run, use service virtualization to create mocks that simulate the behavior of dependent services. This allows you to test specific integration scenarios in isolation, such as how your application handles an error response from an external API.
- **Adopt contract testing** – To avoid the brittleness of traditional integration tests, contract testing is a powerful shift-left strategy. Instead of testing live services together, this technique validates that each service adheres to a shared, version-controlled contract that defines the API's expected requests and responses. This allows teams to develop and deploy their services independently with high confidence that they will not break integrations. Contact-testing tools, such as Pact, are the industry standard for implementing this consumer-driven approach.
- **Automate in CI/CD pipelines** – All integration tests, especially fast-running contract tests, should be fully automated and integrated into the CI/CD pipeline. This helps you detect any integration failures with every code change.

End-to-end testing

End-to-end testing validates the entire application workflow from the user's perspective. It makes sure that all the modular microservices work together correctly. This is crucial for catching integration issues that unit tests miss.

One of the key challenges for end-to-end testing is the non-deterministic nature of LLM responses. These make traditional end-to-end testing unreliable because traditional testing relies on exact-match assertions. Additionally, factors such as response latency and the complexity of multi-turn conversational flows add to the difficulty.

The following are recommended practices for end-to-end testing:

- **Focus on workflow, not exact output** – Instead of asserting that the LLM produces a specific sentence, end-to-end tests should verify that the workflow completes successfully, that the response is structurally correct (such as valid JSON), and that its content is semantically relevant to the original query.
- **Use realistic data** – End-to-end tests should mimic real-world user scenarios by using concrete and realistic test data, including edge cases and potential error conditions.
- **Automate the full flow** – Use UI automation frameworks to simulate user interactions from the frontend through the entire backend stack.
- **Handle asynchronicity and latency** – Tests must be designed to handle variable response times from LLM APIs. Include appropriate waits and timeouts to prevent flakiness.

Offline evaluation

The offline evaluation process that began in the PoC stage is now formalized and scaled in preproduction. *Offline evaluation* is a method that allows you to assess the performance of an AI system by using historical data. The small high-quality dataset of prompts and ideal responses is expanded into a comprehensive, version-controlled evaluation dataset that includes common use cases, known edge cases, and adversarial inputs. This suite becomes the benchmark for automated quality assessment.

The most scalable approach for automated evaluation is using a powerful LLM as the judge. The judge is given the original prompt, the application's response, and a scoring rubric. It then returns a structured score and a rationale. If the same LLM judge has been used during PoC stage, the same LLM judgement instructions and prompts can be reused in the preproduction stage to provide the

same level of evaluation metrics as the PoC stage. For more information, see Evaluation strategies and metrics in the PoC chapter of this guide.

Integrate offline evaluations into your CI/CD pipeline. These automated evaluations are a critical quality gate in the CI/CD pipeline. The pipeline is configured to fail the build if quality scores drop below a predefined threshold, which automatically prevents regressions. Available frameworks, such as pytest-evals and the LangSmith pytest integration, are designed to bring this AI-centric evaluation into standard software engineering workflows.

Online evaluation

Online evaluations are validations with real users. No matter how thorough offline evaluations are, the ultimate validation comes from real user interactions. Controlled rollouts are essential techniques for de-risking the final step into production by testing on live traffic.

The following are recommended techniques for offline evaluations, and you can use these techniques in both preproduction and production environments:

- **A/B testing** – This technique is used to compare two or more versions of a specific component, typically a prompt or model configuration, on key business metrics. For example, a small percentage of users might receive responses generated by a new prompt while the rest use the stable version of the prompt. By analyzing metrics like user feedback scores, latency, and cost for each group, teams can make a data-driven decision about which version performs better. For more information, see A/B testing of LLM prompts in the Langfuse documentation.
- **Canary releases** – This is a broader strategy for deploying an entirely new version of the application. Initially, only a small fraction of traffic, such as 5%, is routed to the new version. The team closely monitors key metrics in real-time. If the performance remains stable, traffic is gradually increased. If issues arise, traffic is immediately rolled back to minimize the user impact. This makes canary testing an essential safety net for deploying unpredictable LLM updates. For more information, see Canary testing for LLM apps (Portkey).

Human-in-the-loop evaluation

The evaluation lifecycle is a continuous loop, not a one-time check. The feedback collected from real users during online testing is the fuel for this loop. Collecting feedback is a critical part of *human-in-the-loop* reviews. The application must be designed with mechanisms to collect both explicit feedback and implicit feedback. Examples of *implicit feedback* are thumbs up and thumbs

down ratings and user comments. Examples of *explicit feedback* are user actions to copy the response or pose a rephrased query.

This feedback is not just for dashboards. The most valuable interactions, especially user-reported failures, should be triaged and converted into new test cases. These are then added to the version-controlled evaluation dataset, which continuously hardens the automated test suite against known, real-world failures. This creates a virtuous cycle where production usage directly improves the quality and robustness of future versions. For more information, see Driving continuous improvement through data and feedback loops in this guide.

Security controls for the preproduction stage

The preproduction stage represents a critical transition where security and governance must evolve from theoretical concepts to fully implemented, tested, and automated realities. This phase focuses on hardening applications through comprehensive testing and implementing production-ready security controls that bridge the gap between experimental PoC and enterprise deployment. With the intended system design now established, you can complete comprehensive threat modelling to identify additional attack vectors and security requirements that might not have been apparent during the initial PoC stage.

Enhance guardrails beyond the basic protections that you established in development. Expand from simple prompt injection filters to include comprehensive content filtering, output validation, and context-aware guardrails that address identified threat modelling and business needs. For more information, see Amazon Bedrock Guardrails enhances generative AI application safety with new capabilities (AWS blog post). Adopt immutable infrastructure patterns, where system components are replaced rather than modified in place. This reduces the risk of unauthorized changes and promotes consistent, reproducible deployments.

The preproduction stage introduces advanced security patterns that might have been too restrictive for PoC development, such as Zero Trust principles. Align organizational security and compliance standards during preproduction to reduce potential redevelopment work. For more information, see Compliance validation for AWS Security Hub CSPM. It is also recommended that you evaluate against the AWS Well-Architected Framework.

In the preproduction stage, security and governance must transition from theoretical concerns to fully implemented, tested, and automated realities. For any enterprise, the deployment of a generative AI application is contingent on demonstrating that it is secure, compliant, and trustworthy. This requires a defense-in-depth approach that includes multi-layered guardrails, offensive security testing, and robust governance frameworks.

Implementing multi-layered guardrails

In the context of generative AI, *guardrails* are a system of programmatic controls that are designed to enforce policies on the inputs and outputs. They are essential for protecting data privacy, maintaining regulatory compliance, preventing misuse, and aligning the model's behaviour with the specific business context. Effective guardrails are not a single feature but an interlocking system of controls that operate at different points in the application's data flow. For example, you can use Amazon Bedrock Guardrails to detect and filter harmful content.

The following are the types of recommended guardrails:

- **Input guardrails** – These controls scan and analyze user prompts before they are sent to the LLM. Their purpose is to detect and block malicious or inappropriate inputs. Key functions include the following:
  - **Detection of prompt injection attacks** – *Prompt injection attacks* involve manipulating prompts to influence LLM outputs, with the intent to introduce biases or harmful outcomes. For more information, see Prompt engineering best practices to avoid prompt injection attacks on modern LLMs (AWS Prescriptive Guidance) and Detect prompt attacks with Amazon Bedrock Guardrails (Amazon Bedrock documentation).
  - **PII detection** – Scan for PII data (such as names, phone numbers and social security numbers) to prevent sensitive data from being processed by the LLM. For more information, see Remove PII from conversations by using sensitive information filters in the Amazon Bedrock documentation.
  - **Denied topic filtering** – Block queries that fall into predefined restricted topics. For example, a customer service bot for a bank should be configured to refuse to give investment advice. For more information, see Block denied topics to help remove harmful content in the Amazon Bedrock documentation.
- **Output guardrails** – These controls scan the LLM's response before it is sent back to the user. Their purpose is to make sure that the generated content is safe, accurate, and compliant. Key functions include the following:
  - **Guardrails to filter harmful content** – Detect and filter content related to hate speech, violence, or other toxic categories, based on configurable thresholds. For more information, see Configure content filters for Amazon Bedrock Guardrails in the Amazon Bedrock documentation.
  - **Contextual grounding checks** – For RAG applications, this guardrail compares the generated response against the source documents that were provided as context. If the response contains

information that contradicts or is not supported by the source data, the guardrail can detect or block it. For more information, see Use contextual grounding check to filter hallucinations in responses in the Amazon Bedrock documentation.

- **PII masking** – Masking acts as a final check to make sure that the LLM has not inadvertently included any sensitive information in its response. For more information, see Remove PII from conversations by using sensitive information filters in the Amazon Bedrock documentation.

For consistency and manageability, these guardrails should be implemented centrally, often as a component within the AI gateway. This helps you apply policies uniformly across all applications and services. You can implement guardrails through policy-as-code frameworks (such as the Open Policy Agent), specialized commercial or open source guardrail tools (such as Amazon Bedrock Guardrails and NVIDIA NeMo Guardrails), or custom-built logic for highly specific needs. For more information, see GenAI Guardrails: Implementation & Best Practices (Lasso blog post).

When choosing a solution, organizations face a trade-off between commercial and open source guardrails. Commercial platforms generally offer faster implementation, access to specialized security expertise, and often come with compliance certifications. This makes them a good choice for enterprises that need to move quickly and need robust protection. Open source solutions provide maximum transparency and control, which is ideal for organizations with deep in-house security expertise and unique requirements. However, these solutions demand significant internal resources to implement, maintain, and continuously update against emerging threats. For more information, see How we're thinking about Generative AI: Proprietary vs Open Source (Medium blog post). In practice, some organizations adopt a hybrid solution of both commercial and open source guardrails to meet their requirements.

Adversarial testing and red teaming

If guardrails are the defensive walls of the application, red teaming is the offensive force that is designed to test their strength. *Red teaming* is a form of ethical hacking where security professionals or automated tools simulate the attacks of a malicious actor to proactively discover vulnerabilities before they can be exploited in the production environment.

This process should not be random; it must be a systematic and repeatable effort that aligns with a recognized framework. The OWASP Top 10 for Large Language Model Applications (OWASP) has become the industry standard for categorizing the most critical security risks that face these systems. The preproduction red teaming effort should be structured to test for these specific vulnerabilities.

The following are key areas to test, based on OWASP top ten:

- **LLM01: Prompt injection** – The red team should employ a variety of techniques (direct, indirect, and jailbreaking) to attempt to override the system's core instructions and cause it to perform unintended actions, such as ignoring previous commands or revealing confidential information.
- **LLM02: Sensitive information disclosure** – Testers should craft prompts designed to trick the model into revealing sensitive data that may have been present in its training set or that exists in the current conversational context.
- **LLM04: Data and model poisoning** – For RAG systems, this is a critical test. The red team should attempt to inject malicious or false information into the knowledge base, such as by uploading a compromised document, to see if they can *poison* the system and corrupt its future answers on that topic.
- **LLM07: System prompt disclosure** – A common attack is to try to extract the system prompt itself because it often contains proprietary business logic, instructions, or context that the developers do not want exposed. The red team should test various extraction techniques.

This testing should be automated wherever possible by using tools such as Promptfoo, which can run suites of adversarial tests. These automated scans should be integrated directly into the CI/CD pipeline to make sure that every new version of the application is tested for these critical vulnerabilities before deployment.

Establishing governance and compliance

Finally, the application must be integrated into the enterprise's broader governance and compliance framework. Implement the following in your organization:

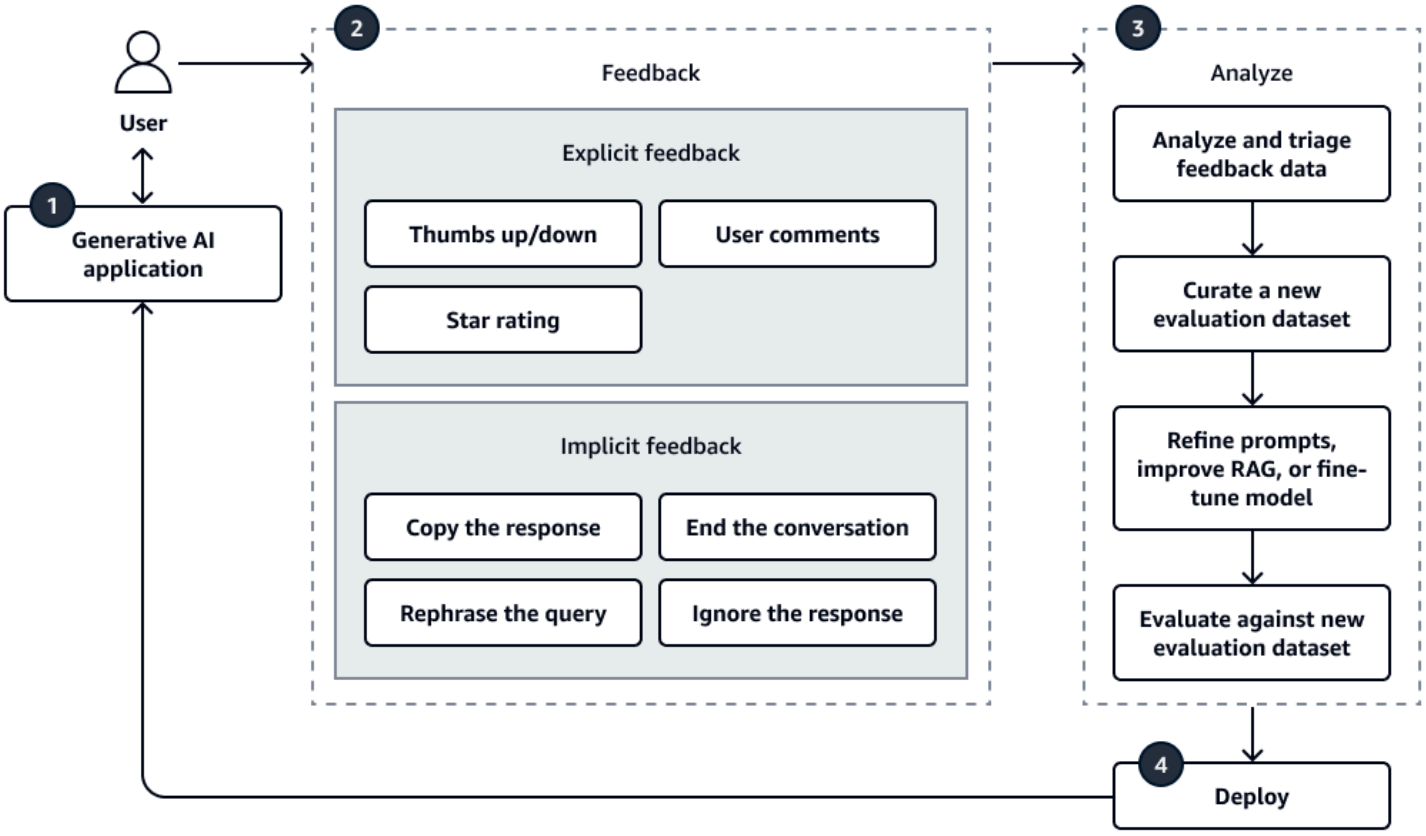
- **Data governance** – All data handling processes, from ingestion into the RAG pipeline to the storage of conversation logs, must adhere to the organization's data governance policies and relevant regulations, such as General Data Protection Regulation (GDPR) or Health Insurance Portability and Accountability Act (HIPAA). This includes policies for data classification, retention, and deletion. The AI gateway can play a role in enforcing data sovereignty by automatically routing requests from users in a specific geography to model endpoints that are hosted in a compliant AWS Region.
- **Access control** – Robust role-based access control (RBAC) must be implemented. This makes sure that users and services can only access the data and functionalities for which they are explicitly authorized. The application's authentication and authorization system should be integrated with the enterprise's central identity provider to consistently enforce policies.

- **Audit trails** – The observability system must be configured to provide a comprehensive and immutable audit trail. This log should capture all prompts, responses, tool calls, and actions taken by the agent. This detailed record is non-negotiable for compliance purposes and is invaluable for forensic investigation in the event of a security incident.

Driving continuous improvement through data and feedback loops

A generative AI application is not a static artifact that is finished upon deployment. The real world is dynamic; new information becomes available, user expectations evolve, and new failure modes emerge. Without a mechanism to adapt to this changing environment, the quality and relevance of any generative AI application will inevitably degrade over time. The preproduction stage is where you must design and build a continuous learning architecture that learns from data and feedback loops.

The following diagram shows a feedback loop that drives continuous improvement.



The diagram shows the following continuous improvement cycle:

1. The user interacts with the generative AI application

2. The user provides implicit or explicit feedback through the generative AI application.
3. The application team analyzes the feedback, updates the evaluation dataset, and improves the prompt, system, or model.
4. The application team deploys the new version of the generative AI application.

To build this continuous improvement loop, you must design a robust feedback system, build a data collection pipeline for the user feedback, and then iterate the application based on that feedback.

Designing robust feedback systems

A robust feedback architecture captures signals from multiple sources as either explicit or implicit feedback.

Explicit feedback involves designing UI components that actively and directly solicit feedback from users. This is the most direct signal of quality. Common mechanisms include:

- Thumbs up or down buttons for each generated response
- A star rating system, such as 1-5 stars
- A **report issue** button that opens a form where users can categorize the problem (such as *inaccurate*, *harmful*, or *irrelevant*) and provide a brief explanation of the issue

Implicit feedback involves capturing user behaviors that imply satisfaction or dissatisfaction, without requiring the user to take an explicit action. These signals can be noisy but are valuable at scale. Examples include:

- (Positive signal) The user copies the generated response to their clipboard.
- (Positive signal) In a conversational agent, the user ends the conversation after receiving an answer.
- (Negative signal) The user immediately rephrases and resubmits their query after getting a response.
- (Negative signal) The user ignores the generated response and continues scrolling or navigates away.

All of this feedback data, both explicit and implicit, must be logged. Critically, it must be tied back to the unique trace ID of the specific interaction that generated it. This allows engineers to

correlate a piece of feedback directly with the full context of the request, including the prompt, the model's response, and the tools that were called.

Architecting the feedback collection pipeline

Designing a robust feedback system involves meticulous planning of how feedback is collected, tracked, and stored. The following are common aspects in a feedback collection pipeline:

- **Frontend integration and design** – The user interface, as the primary feedback channel, should minimize friction by offering simple tools such as rating buttons with optional comment fields. Feedback requests must appear at natural stopping points to avoid disrupting user flow. Responsible AI design also emphasizes that users should remain in control. AI involvement should be clearly disclosed. This means that users should know when content, suggestions, or responses are generated by AI rather than humans, and users should have the ability to dismiss or revert AI-generated content.
- **Backend storage and request tracking** – All user interactions should be continuously collected, tracked, and logged. Linking feedback to unique request IDs enables detailed analysis by connecting user satisfaction to specific model outputs. Storing feedback with metadata, such as user ID and timestamp, is essential for thorough troubleshooting, performance monitoring, and extended evaluation.
- **User behavior analytics** – Beyond explicit feedback, analyzing logs and using machine learning platforms can provide deeper insights into user patterns. It can also help you predict future trends, identify opportunities and risks, and drive continuous system improvement.

Analyzing the feedback and improving the application

Collecting feedback is useless if it sits dormant in a database. The final and most important part of the architecture is the automated workflow that closes the loop by turning raw feedback into concrete application improvements. This workflow typically involves the following stages:

- **Collect and aggregate** – All feedback data is collected from the application logs and aggregated in a centralized data warehouse or analytics platform.
- **Analyze and triage** – Dashboards and automated analysis are used to identify trends and patterns in the feedback. Which types of queries are receiving the most negative interactions? Is there a particular document in the RAG knowledge base that is frequently associated with responses marked as inaccurate?

- **Curate for evaluation** – The most valuable pieces of feedback, especially user-reported failures with clear explanations, should be automatically triaged and converted into new test cases. These new prompt-and-expected-outcome pairs are added to the versioned evaluation dataset, hardening the test suite against known failures.
- **Retrain or refine** – The aggregated feedback data provides a powerful signal for improvement. It can be used to guide prompt engineering efforts or to identify gaps in the RAG knowledge base that need to be filled. In more advanced scenarios, you can use it to create a preference dataset for fine-tuning the model by using techniques like reinforcement learning from human feedback (RLHF). You might go back to the PoC stage with the extended datasets for evaluation or fine-tuning.
- **Re-evaluate and deploy** – After a potential fix has been implemented, the improved version of the application is tested against the newly updated evaluation dataset. This confirms that the original issue has been resolved and that the change has not introduced any new regressions.

The quality of a generative AI application over its lifetime is a direct function of the quality and speed of its feedback loop. A system with no feedback loop is doomed to obsolescence. A system with a slow, manual feedback loop will struggle to keep pace with the changing world. A system with a fast, automated, and comprehensive feedback loop will continuously learn and adapt, maintaining or even improving its quality and value over time. Therefore, the feedback architecture is not a nice-to-have feature for post-launch improvement. It is a core architectural component that determines the long-term viability of the product. You must be designed built it with the same rigor as the core application logic during the preproduction stage.

Advancing your generative AI application to production

The culmination of the preproduction stage is a formal go or no-go decision for production deployment. This decision should not be based on gut feelings, stakeholder pressure, or looming deadlines. It must be a data-driven decision based on whether the application has met a predefined, objective, and comprehensive set of exit criteria. Successfully navigating the preproduction stage is the single most important factor in determining whether a generative AI project will deliver tangible business value or become another stalled statistic.

The journey requires architects to design for the harsh realities of production by embracing modular, cloud-native patterns and centralized governance. It requires engineers to build a robust GenAIOps framework that automates the entire application lifecycle, including full-stack versioning, deep observability, and CI/CD pipelines. To build confidence in a non-deterministic

system, you need a multi-layered testing strategy that combines traditional methods with novel automated evaluations and controlled rollouts. And it requires an unwavering commitment to security, with implemented guardrails, offensive red teaming, and auditable governance. Finally, it requires a continuous improvement loop that improves the application based on explicit and implicit user feedback.

By establishing these capabilities and holding the project accountable to a strict set of metric-driven exit criteria, organizations can transform a fragile prototype into a resilient, scalable, and trustworthy enterprise application. A go decision at this stage signifies that the organization has high confidence in the solution's ability to deliver sustained value and that it is ready to commit the resources for a full production rollout.

GLOE stage 3: Production, deployment, and continuous operation of generative AI applications

Moving a generative AI application into production marks a critical transition to real-world value delivery. This stage requires robust systems for deployment, monitoring, and continuous improvement to ensure the application remains reliable, secure, and aligned with business objectives. Rather than viewing production deployment as a destination, organizations must approach it as the beginning of an ongoing optimization journey that includes continuous monitoring. The goal is to make sure that the AI system stays relevant and produces valid results, even as user behaviors evolve, new data patterns emerge, and underlying data distributions gradually drift over time.

The production environment introduces unique challenges that demand specialized operational frameworks. These include detecting and responding to model drift, gathering and acting on user feedback, maintaining security at scale, and establishing governance protocols that address the distinct risks of generative AI systems. Success in this stage requires careful orchestration of technical, operational, and organizational elements to create sustainable systems that consistently deliver business value while managing potential risks.

This chapter is organized into two main sections:

- Delivering and sustaining the value of a generative AI application – The first section focuses on how to deliver sustained value from generative AI applications in production and how to define success through key business objectives.
- Performance monitoring and continuous improvement for generative AI applications – The second section describes how to monitor and improve production generative AI applications through performance monitoring, drift detection, feedback loops, and security controls.

Delivering and sustaining the value of a generative AI application

The transition from the preproduction stage to the production stage marks a fundamental shift in the primary objective of a generative AI initiative. The preproduction stage is designed to answer critical questions of operational readiness, such as whether the solution is reliable under a realistic

load, whether it is secure and compliance, and whether it is safe and viable for enterprise use. Preproduction is a phase of hardening, testing, and derisking.

This section establishes the strategic context for the production stage. It shifts the focus to the primary business imperatives for a live application: delivering consistent user value, achieving measurable strategic goals, and demonstrating a clear ROI.

This section contains the following topics:

Validating the value of a generative AI application

Once an application enters the production stage, multi-faceted stakeholder requirements collectively determine the success of the generative AI application. For example:

- Business executives must validate that the solution delivers on the promised ROI.
- Business units need proof that the application fulfills its operational requirements within budgetary constraints.
- Security, legal, and compliance teams need verifiable audit trails, robust governance, and adherence to evolving regulatory frameworks.
- Technical teams need to monitor system health indicators to confirm infrastructure scalability and reliability, and they need to track model performance.

Entering production is not a project conclusion. It is the beginning of a strategic business asset lifecycle that requires synchronized governance across business functions. The GLOE framework posits that production is not a static state but a dynamic process of value delivery, optimization, and evolution. This continuous lifecycle is essential because, unlike deterministic software, the performance and relevance of a generative AI application can degrade over time due to factors like data drift, concept drift, and shifting user expectations.

Therefore, the objective is to move beyond simply deploying a generative AI application to actively managing it as a living system that must consistently prove its worth. This requires a robust operational framework for monitoring performance, gathering feedback, and driving iterative improvements to make sure that the application remains aligned with strategic business goals and delivers a quantifiable return on investment.

Defining success through business outcomes and KPIs

A core challenge in generative AI projects is the potential disconnect between the technical metrics tracked by engineering teams and the business value expected by leadership. To bridge this gap, the *evaluation framework* introduced in the PoC stage must evolve into a comprehensive, live KPI-monitoring framework. This framework makes sure that every technical metric is traceable to a meaningful business outcome. This creates a shared language across different functional teams.

The cost of operating a generative AI application is not a fixed, one-time expense. It is a dynamic variable that is influenced by factors such as API token consumption, infrastructure scaling to meet demand, and the computational costs associated with maintenance activities like model fine-tuning. Similarly, the value delivered by the application is not static. It is affected by user adoption rates, shifts in user behavior, the potential degradation of model performance over time, and model changes for cost or performance efficiencies.

Consequently, ROI cannot be treated as a static calculation that is performed at launch. It must be managed as a dynamic KPI that is continuously tracked and visualized on a dashboard. This transforms ROI from a historical justification for a project into a real-time operational health indicator for business leaders. An *ROI dashboard* should be considered a primary artifact of the production stage. It should integrate financial metrics (such as cost per interaction and total infrastructure spend) with business value metrics (such as hours saved, revenue lift, and CSAT score improvements). This dashboard provides a clear, ongoing justification for the application's existence, and it can guide future investment decisions.

Performance monitoring and continuous improvement for generative AI applications

Performance monitoring and continuous improvement are essential components for maintaining generative AI applications in production environments. As these systems operate in dynamic conditions with evolving user behaviors and data patterns, robust monitoring frameworks and improvement mechanisms are crucial for ensuring sustained reliability, effectiveness, and safety. This section provides a comprehensive examination of the key operational requirements and technical approaches for managing generative AI applications at scale.

The monitoring and improvement framework presented in this chapter is structured around three fundamental pillars: application health monitoring, business metrics tracking, and model quality assessment. Special attention is given to detecting and addressing model drift, which is a

critical challenge in production AI systems. Each pillar encompasses specific metrics, measurement methodologies, and intervention mechanisms that are designed to address the unique challenges posed by generative AI systems. Through this multi-faceted approach, organizations can maintain comprehensive oversight while implementing data-driven improvements to their production systems.

This section details the technical implementation of monitoring systems, drift detection mechanisms, feedback loops, and security controls. It provides practical guidance for engineering teams that are responsible for production generative AI applications, and it covers both the architectural considerations and operational procedures required for enterprise-grade deployments. Systematic, data-driven approaches can help you maintain system reliability while enabling continuous improvement through structured evaluation and iteration processes.

This section contains the following topics:

Pillars of monitoring generative AI application performance in production

Effective performance monitoring in the production stage helps you make sure that the system remains reliable and effective. A well-rounded monitoring framework includes monitoring across the following three pillars:

- **Application and system health** – This pillar helps you validate that the generative AI application is highly available and cost efficient. It focuses on the operational stability, scalability, and runtime performance of the application and support infrastructure. Some example key metrics include:
  - **Latency** – Speed of response delivery
  - **Throughput** – Volume of requests

  - **Uptime and reliability** – System availability, error rates, and service-level agreement (SLA) adherence
  - **Resource utilization** – CPU, GPU, memory, and bandwidth consumption
  - **Cost efficiency** – Infrastructure and API usage costs
- **Business and user-interaction health** – This pillar helps you evaluate whether the application meets business objectives. This can include tracking adoption, customer satisfaction, and measurable impacts on the business process. Example key metrics include:
  - **User engagement** – Track active user count, repeat usage, task completion rates.
  - **Customer satisfaction** – Monitor net promoter score (NPS) and qualitative feedback trends.
  - **Business impact** – Track productivity improvements, cost savings, revenue growth, or task automation efficiency.
  - **Compliance and risk indicators** – Adhere to governance requirements, such as logging actions related to data access, model changes, and production deployments. Track regulatory compliance, such as tracking use of PII and verifying consent. Investigate and resolve policy violations, such as improper model use or ethics complaints.
- **Model and AI quality health** – This pillar helps you assess the accuracy, consistency, and fairness of the model. It helps you make sure that the output remains relevant, aligned with the intended outcome, and free from degradation, hallucinations, or ethical issues. Key metrics include the following:
  - **Accuracy and relevance** – Alignment of output with user prompts
  - **Hallucination rates** – Frequency of generating false or misleading information
  - **Bias and fairness** – Monitoring for skewed or harmful outputs
  - **Model drift detection** – Identifying any shift in the model effectiveness due to changing inputs
  - **Explainability and traceability** – Attributing outputs to a specific model version, prompt, or knowledge source
  - **Prompt and knowledge base versioning** – Measuring the impact of prompt engineering or a RAG dataset over time

Production monitoring underpins a robust system with three main key components: capture, alerts, and response. Monitoring should provide real-time insights into the health of the application, actionable alerts, and (ideally) automation to react to these alerts.

When implementing the metric or log capture system, you need to consider the following:

- Which metrics and logs need to be captured
- How to capture this information
- How frequently to record it
- When to aggregate metrics or summarize logs
- Where to store the captured information
- How to access the captured information

While having metrics, logs, and traces is useful, you often need to sift through a lot of information to find and detect a problem. Thus, it is important to define and create actionable alerts that align to your business objectives and technical requirements. Use rule-based or ML-based mechanisms to detect errors or anomalies in the system, and confirm that these alerts are directed to the relevant stakeholders with an actionable outcome.

The final component of a monitoring system focuses on how to respond to these alerts. The response should be systematic and, ideally, automated. This involves designing runbooks that attach to each alert in order to help teams effectively perform root-cause analysis. Alternatively, you can create codified runbooks that automate the investigation and resolution process.

Detecting drift in production applications

In the context of LLMs, *drift* refers to the gradual degradation of its performance over time. This is typically caused by changes in data distributions or user behavior that the model was not originally trained on. Like ML models, the output of generative AI models is dictated by their input. When the input changes, the output from these models can drift from the original intent. This can lead to a degradation of the generative AI application. In extreme cases, it can produce detrimental outcomes in terms of safety, legal, or monetary losses. A proactive drift-monitoring framework is essential for maintaining quality and reliability.

Types of drift in LLMs

Drift can be categorized into two main types, both of which can affect performance: data drift and concept drift.

Data drift refers to a statistical change in the input data the model receives. In LLMs, this is most effectively measured as a shift in the distribution of input prompt embeddings. Data drift occurs when the topics, questions, or language style of user prompts in production begin to differ

significantly from the data the model was trained or last evaluated on. For example, a customer service bot for a mobile phone company might experience data drift after the launch of a new flagship phone because the distribution of user queries shifts to this new topic.

Concept drift refers to a more subtle change in the underlying relationship between inputs and the desired outputs. It occurs when user expectations change or when the meaning of concepts evolves over time. For example, in a financial analysis application, the factors that define a "good investment" might change due to new market conditions. The user prompts might look statistically similar (which means that the data drift is low), but the desired answer has changed (which means the concept drift is high). Detecting concept drift is significantly more challenging. It often involves monitoring downstream business metrics and user feedback rather than direct statistical tests on inputs.

Data drift detection

A robust framework for detecting data drift in LLM embeddings should be multi-layered. It should combine efficient statistical methods for initial detection with more sophisticated semantic analysis for interpretation.

Layer 1: Statistical drift on embeddings

This layer serves as the automated, first line of defense. It follows this process:

1. **Establish a baseline** – Capture a representative sample of prompt embeddings from a stable period, such as the first month of production, to serve as the reference distribution.
2. **Monitor production data** – In real-time or in batches, capture the embeddings of incoming production prompts to create the current distribution.
3. **Compare distributions** – Use statistical tests to quantify the distance between the reference and the current distributions. Commonly used drift metrics, such as the Kolmogorov–Smirnov (KS) test are less effective for generative AI use cases because of the multi-dimensional nature of the LLM embeddings. It's therefore important to use statistics that perform better at measuring drift changes in embedding spaces, such as Wasserstein distance.
4. **Alert if a threshold is breached** – If the calculated distance exceeds a predefined threshold, an alert is triggered. This alert indicates that significant data drift has occurred.

For more information about statistical detection of drift, see the following resources:

- Feature attribution drift for models in production (Amazon SageMaker AI documentation)

- Monitor embedding drift for LLMs deployed from Amazon SageMaker JumpStart (AWS blog post)
- How A&E Engineering Uses Serverless Technology to Host Online Machine Learning Models (AWS blog post)
- Detecting data drift using Amazon SageMaker AI (AWS blog post)
- Learning High-Density Regions for a Generalized Kolmogorov-Smirnov Test in High-Dimensional Data (NeurIPS Proceedings)
- Kolmogorov-Smirnov (KS) (Amazon SageMaker AI documentation)

Layer 2: Semantic drift using an LLM

A statistical alert indicates that a drift has happened, but it doesn't indicate why. To gain actionable insights, you can use the LLM-as-a-judge approach to analyze and classify the nature of the drift:

1. **Sample the drifted data** – When a statistical drift alert is triggered, collect a sample of the prompts from the period that caused the alert.
2. **Perform a semantic analysis and classify the drift** – Use a judge LLM with a carefully engineered prompt to compare the drifted prompts to a sample from the reference baseline. The judge's task is to categorize the nature of the change. For example, it might be prompted to classify the primary reason for the drift as "Emergence of a new topic," "Shift in user intent," "Increase in query complexity," or "Change in language style."
3. **Review the results** – The results of this classification provide the human team with a clear understanding of the drift's root cause. This classification guides the subsequent improvement actions.

For more information about using an LLM to gain a semantic understanding of detected drift, see the following resources:

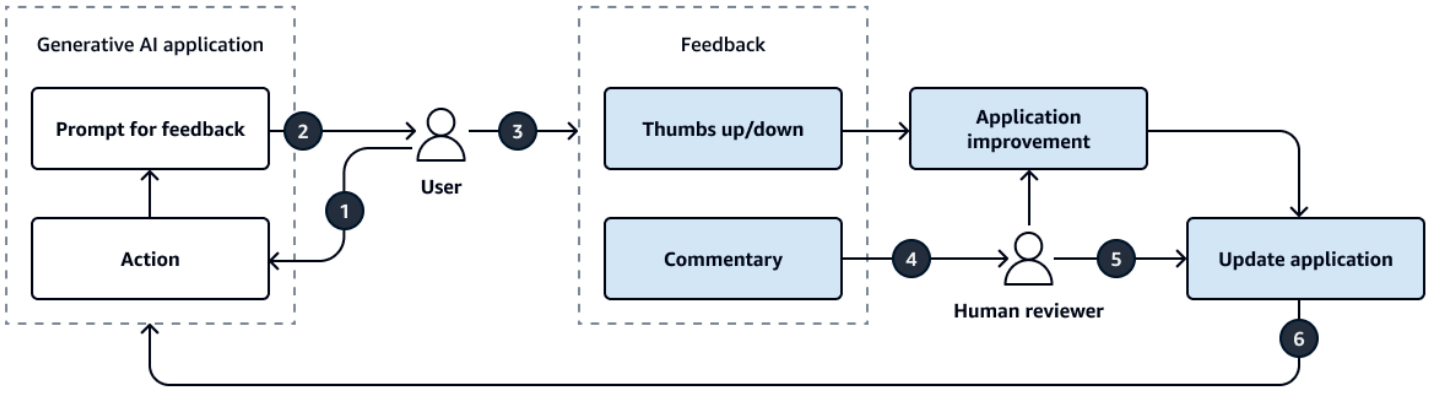
- Joint Detection of Fraud and Concept Drift inOnline Conversations with LLM-Assisted Judgment (Arxiv)
- LLM-as-a-judge: a complete guide to using LLMs for evaluations (Evidently AI)

Architecting the production feedback loops

Balancing generative AI application autonomy with human oversight is central to good agent design. Generative AI applications should be able to work autonomously because independent operation is what makes it valuable. However, humans should retain control over how they achieve their objectives, especially for high-stake decisions. A foundation of continuous improvement is a well-architected data pipeline that captures and centralizes all forms of user feedback. This feedback is the most valuable source of ground truth data about the application's real-world performance. This section provides several approaches to involve humans in the generative AI application to validate that the agents are working within the defined operational parameters and are safe and reliable.

User feedback loop

Teams comprehensively test through robust testing and the use of synthetic events to simulate real-world scenarios. However, it can be difficult to emulate some of real-world behaviors, or certain scenarios might be missed in testing. For an interactive generative AI system, it is recommended that you implement a feedback mechanism that captures user feedback when the application is not functioning as intended. This can be as simple as a binary good or not-good response. It could also be a more comprehensive open-text system where users can provide verbose feedback. The Driving continuous improvement through data and feedback loops section of this guide discusses how to implement a comprehensive feedback strategy that captures explicit and implicit user feedback. The same concepts also apply to the production stage. The following image shows a feedback loop in a production stage.



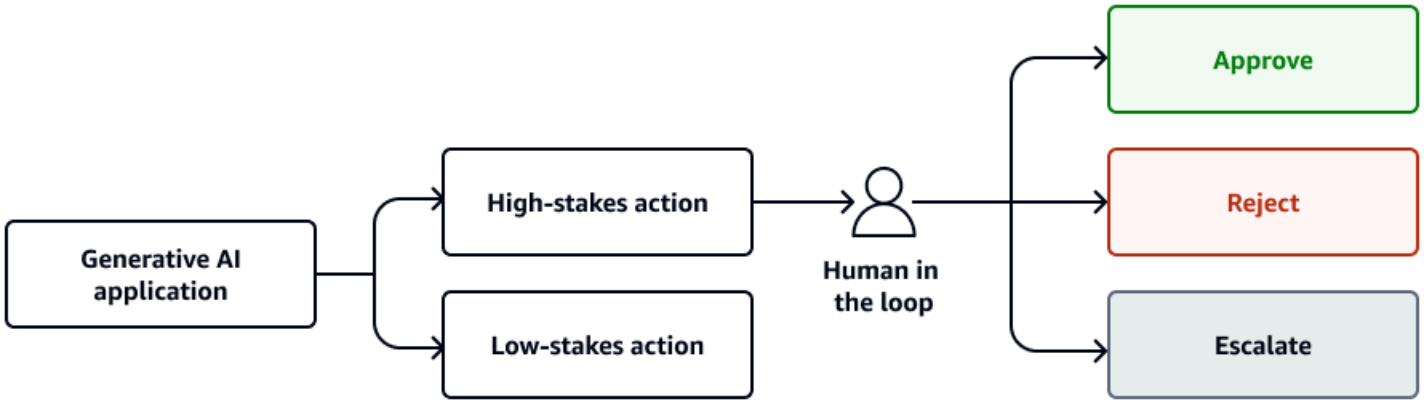
The diagram shows the following workflow:

1. The user performs an action in the generative AI application.
2. The generative AI application prompts the user for feedback.

3. The user provides feedback in the form of a thumbs-up or thumbs-down response or in the form of commentary.
4. A human reviewer evaluates the feedback responses.
5. The human reviewer fine-tunes the model and updates the generative AI application.
6. The updated application is deployed to the production environment.

Human in the loop

Generative AI applications can be non-deterministic. When these applications are making autonomous decisions or actions, they can make incorrect decisions. It's imperative that you design for these failures through the implementation of safety guardrails. One approach to balance the autonomy of the generative AI application is to allow the application to make automated actions only in well-defined and low-risk scenarios. You incorporate a human-in-the-loop mechanism for high-risk or unfamiliar scenarios that are not previously covered in testing scenarios. The following diagram shows how you can include a human in the loop to review high-stakes actions.



Feedback data pipeline and schema

Collecting feedback is only useful if it is structured and linked to the interaction that generated it. A robust feedback system requires a well-defined data pipeline and a corresponding data schema that makes sure that every piece of feedback can be traced back to its origin.

The feedback pipeline

The architecture should consist of a dedicated, centralized service that ingests all feedback events from the application frontend and backend. This service is responsible for validating the data and storing it in a structured format, such as a data warehouse or a dedicated analytics database.

Essential data schema

The power of the feedback loop comes from correlating feedback with the full context of the interaction. Therefore, the schema for storing feedback must link each feedback event to the complete application trace. An example schema includes the following fields:

- `feedback_id` – A unique identifier for the feedback event.
- `trace_id` – A foreign key that links to the unique ID of the end-to-end application trace. This is the single most important field. It allows retrieval of the exact prompt, retrieved documents, model response, latency, model version, prompt template version, and any tool calls associated with the interaction.
- `timestamp` – The time the feedback was submitted.
- `user_id` – An identifier for the user providing the feedback.
- `feedback_type` – An enum that indicates the source of the feedback, such as `explicit_thumb`, `implicit_copy`, or `explicit_comment`.
- `feedback_value` – The value of the feedback, such as a 1 for a thumbs up, a 0 for a thumbs down, or a numeric score from 1–5.
- `feedback_comment` – An optional string for storing free-text user comments.

This structured schema transforms feedback from a collection of isolated opinions into a powerful dataset that you can query for debugging, analysis, and the creation of new evaluation sets.

Turning insights into improvements in generative AI applications

Production monitoring of generative AI applications generates valuable data about system performance, user interactions, and failure patterns. This section describes systematic approaches for converting monitoring insights into concrete improvements. It examines key intervention mechanisms for handling application failures, methods for conducting root-cause analysis, and processes for implementing improvements through the GLOE lifecycle. The patterns and workflows presented help teams to establish robust feedback loops between production observations and system enhancements. This promotes continuous improvement of generative AI applications while maintaining operational reliability.

Intervention mechanisms

Occasionally, a generative AI application might not function as intended. It is important to be able to design for such failures through the implementation of an intervention mechanism. The following patterns are designed to handle such failures.

Pattern 1: Intervention for repeat failures

Imagine that you have a chatbot where the user makes a request for information; however, after several attempts, the user still has not found the right information. The underlying issue could be the information is not available, or perhaps the model is struggling to interpret the user's request. By implementing an intervention mechanism, you can either redirect the request to a human agent who can better assist the user, or you can gracefully fail and send an alert that includes the logs and traces so that an engineering team can investigate. For both mechanisms, it is important to capture logs, traces, and feedback because this information helps engineering continuous improve the generative AI application.

Pattern 2: Intervention for high-risk actions

While using generative AI applications to automate actions can produce large efficiency gains, there are times where some actions can result in irreversible or detrimental outcomes. For example, it might approve a large fund transfer for the wrong item. To address this, you can implement a human-in-the-loop review process to help mitigate such risk. For more information, see Human in the loop in this guide.

Root-cause analysis and loopback to the GLOE lifecycle

When a performance issue is detected, either through a monitoring alert (such as a drift alarm) or a stream of negative user feedback, a systematic workflow needs to be developed to handle this. This process operationalizes the l*ooping cycle* at the heart of the GLOE framework. It makes sure that production insights are methodically channeled back into improvements. The following are the steps to resolve performance issues.

Step 1: Triage and root-cause analysis

The first step is to investigate the *why* behind the failure. Using the trace_id associated with a negative feedback event or a failed evaluation, the team can retrieve the full context of the interaction from the observability platform. The root cause of a failure in a generative AI system might be one of the following:

- **Prompt and orchestration failure** – The issue lies in the software layer of the generative AI application. This could be a poorly constructed prompt template, a flawed reasoning step in a chain-of-thought prompt, or incorrect logic in an agent's decision to use a tool. The underlying LLM and knowledge base might be perfectly capable, but they were given the wrong instructions.
- **Knowledge and retrieval failure** – The issue lies in the data provided to the model. In a RAG system, this is the most common cause of factual errors. The retrieval system might have failed to find the relevant document or retrieved an outdated or incorrect document. Alternatively, the necessary information might not exist in the knowledge base.
- **Core model limitation** – The issue lies with the inherent capabilities of the foundational model itself. Even with a perfect prompt and context, the model might lack the specialized knowledge, reasoning ability, or stylistic nuance to generate the desired output.

Step 2: Looping back to the GLOE lifecycle

The diagnosed root cause dictates the appropriate remediation path, which creates a systematic and efficient loop back to the relevant stage of the GLOE lifecycle. This prevents wasted effort by making sure that the right team uses the right process to fix the problem. For the type of failure, do the following

- **For prompt and orchestration failures, loop back to the development (PoC) stage** – This is fundamentally a software engineering or prompt engineering problem. The failing interaction, which includes the prompt, context, and the user's desired outcome (if available), is packaged as a new, failing test case. You add it to the version-controlled evaluation dataset. Engineers then return to the rapid, iterative experimentation loop of the PoC stage to refine the prompt template or the agent's orchestration logic until the new test case passes.
- **For knowledge and retrieval failures, loop back to the data management (RAG) pipeline** – This is a data problem, not a code problem. The solution does not require developers to change the application. Instead, it is an operational task for data curators or domain experts to update the RAG system's knowledge base. This might involve adding new documents, correcting or updating existing ones, or improving the data processing pipeline. For example, you might adjust the document chunking or embedding strategy.
- **For core model limitations, loop back to the development (PoC) stage for fine-tuning** – This is the most resource-intensive path. If the foundational model itself is the bottleneck, the solution is to enhance its capabilities. This requires collecting a curated dataset of failure cases that are derived from production feedback. Then you initiate a model improvement process by either

selecting a more powerful model or by customizing the model through fine-tuning. Any new model is a major change, and it must be rigorously validated in the experiment environment against the full evaluation dataset before it can be considered for deployment.

Advanced operations for generative AI applications in production

This section helps you consolidate the operational practices that are required to run a generative AI application at an enterprise scale. It builds upon the foundations of security, governance, and deployment that were established in the preproduction stage, and it details how these practices are maintained and evolved in a live, dynamic environment.

Iterative and safe deployment

Fundamentally, the deployment of generative AI applications is like deploying software applications—they can follow common DevOps best practices to reduce deployment risk and drive continuous improvements. A key component in this is deployment automation, which leverages CI/CD pipelines to automate build, testing, and deployment for code, models, and infrastructure. Within the CI/CD pipeline, you should implement release promotion gates to manage the release process with a human-in-the-loop approach. Each release promotion gate should target a specific persona to evaluate the application. This includes, but is not limited to, a technical reviewer to review code quality, a data scientist to review evaluation results, and business application owners to review usability. Human reviewers should also serve as the ultimate promotion gate to evaluate aspects of the application where automated testing can't fully assess. This multi-layered approach makes sure that generative AI applications are thoroughly vetted before reaching users.

Because not all aspects of a generative AI application can be fully replicated in a synthetic environment, it is imperative to use different release strategies to discover these aspects, while limiting the exposure of the new application features. These strategies include the following:

- **Shadow testing** – This is a low-risk strategy for evaluating a new model or major logic change. The new version (the *shadow*) is deployed alongside the current production version. It receives a copy of the same live user traffic. Its responses are generated and logged, but they are not shown to the end-user. The production system continues to serve responses from the existing version. This allows teams to compare the performance of the shadow version against the production baseline for key metrics, such as latency, cost, and AI quality scores. The shadow version operates under real-world conditions without affecting the user experience.
- **A/B testing** – This technique is used to make a data-driven decision between two or more competing versions of a component, typically a prompt or a model configuration. A small,

statistically significant portion of user traffic, such as 5%, is routed to the new version (version B), while the majority remains on the current version (version A). The system then collects and compares business-critical metrics for both groups, such as user feedback scores, task completion rates, or click-through rates. This provides quantitative evidence for which version performs better on the metrics that matter most to the business.

- **Canary deployment** – This is the standard, risk-mitigating strategy for rolling out any significant application update. You initially release the new version to a very small subset of users, which is called the *canary* group. This user group might account for 1–5% of traffic. The team closely monitors a full suite of health metrics, such as system health (latency and error rates), cost, and AI quality. If all metrics remain stable and within their service-level objectives (SLOs), traffic is gradually increased to larger percentages. If any critical metric degrades, traffic is immediately and automatically rolled back to the previous stable version, which contains the impact to a small user base.

CI/CD pipeline for generative AI in production

The CI/CD pipeline is the automated engine that enables these safe deployment strategies. In the production context, its role is to enforce the highest standards of quality and security before any code or configuration can affect users. Key production-stage characteristics of the pipeline include the following:

- **Automated quality gates** – Every proposed change, whether to code, a prompt, or a model configuration, must automatically initiate a pipeline run. The pipeline executes the full, version-controlled evaluation suite. This includes running the application against the high-quality evaluation dataset and scoring its outputs by using an LLM-as-a-judge approach. A build should automatically fail if quality scores, such as faithfulness or relevance, drop below a predefined threshold. This helps prevent quality regressions from ever reaching production.
- **Automated security gates** – The pipeline must also include automated security scans. This includes static analysis of code. Critically, it also includes running a suite of automated adversarial tests to check for common vulnerabilities, such as prompt injection or PII disclosure.
- **Human-in-the-loop approva**l – While the pipeline is automated, the final promotion to the production environment must be gated by a formal human-approval step. This validates that that key stakeholders from product management, information security, and business leadership have reviewed the changes and the results of the automated checks and have provided explicit sign-off for the release. This combines the rigor of automation with the accountability of human oversight.

Enterprise-grade security and governance for generative AI applications

Enterprise deployment of generative AI applications requires robust security controls and governance frameworks that address both traditional cybersecurity concerns and AI-specific vulnerabilities. This section outlines comprehensive approaches for implementing security measures, risk management protocols, and compliance controls specifically designed for generative AI systems. It examines the unique threat landscape these applications face, including prompt injection attacks and model manipulation, along with corresponding mitigation strategies. It also covers regulatory compliance requirements across multiple jurisdictions and presents operational frameworks for implementing responsible AI practices. These components form an essential foundation for maintaining secure, compliant, and ethically sound generative AI applications in production environments.

Production environments demand comprehensive security controls that address the heightened risks associated with live, customer-facing generative AI systems. The production stage represents the culmination of security maturity, where all previous controls are enhanced and supplemented with enterprise-grade protections that are designed for continuous operation under real-world threat conditions. Continuous threat modelling becomes essential at this stage. You should incorporate inputs from the evolving threat environment, model updates, and real user-interaction patterns to identify emerging risks and adapt security measures accordingly.

Security and risk management for generative AI applications

Generative AI applications, being a function of LLMs, prevent an entirely new threat landscape when compared to traditional software, which is susceptible to attack vectors such as SQL injection, cross-site scripting and buffer overflows. Generative AI systems are open to sophisticated adversarial techniques, such as prompt injection attacks, jailbreaking, training data poisoning, model inversion, knowledge extraction, and alignment manipulation. These attacks can subvert intended safeguards.

To mitigate these risks, organizations must implement a comprehensive strategy that is specifically designed for generative AI systems. This strategy includes the following measures:

- **Input prompt filtering** – Deploy adaptive prompt filtering with contextual analysis to defect manipulation attempts while maintaining legitimate functionality.
- **Runtime monitoring** – Implement real-time anomaly detection to identify unusual usage patterns.

- **Permission model** – Implement fine-grain access controls that are tailored to different generative AI functions, tasks, and risk levels.
- **Output safeguards** – Employ multi-layered content moderation with automated scanning. For high-risk scenarios, use a human-in-the-loop approach.
- **Comprehensive observability** – Create audit trails that capture end-to-end interactions. Collect the model version, prompt, response, and user context or metadata.
- **Red teaming** – Conduct adversarial testing with domain experts in AI security to identify vulnerabilities.

Implement shutdown capabilities that provide critical safety mechanisms for high-risk scenarios, including systems that can detect when operations exceed acceptable bounds. These capabilities must be coupled with metrics from guardrail filters and business identified observability markers. You should also consider implementing continuous security posture management through automated scanning and assessment. This helps prevent attackers from exploiting configuration weaknesses. Edge protection controls limit request volumes and filter known threats with rapid rule update capabilities.

The production stage introduces strict no-human-access policies for production data, applications, and infrastructure. All access to production systems should be automated through approved deployment pipelines, with break-glass procedures that require explicit approval and comprehensive logging. This approach minimizes human error and unauthorized access while maintaining operational capability through controlled automation. You must implement robust recovery methods to support remediation of degraded systems within business-acceptable time frames, balancing security response with operational requirements.

Fine-grained access controls and comprehensive audit trails complete the production security posture. Access controls must adapt to the complex interaction patterns that emerge in production environments, where systems typically interact with more services, access more data sources, and serve more users than during earlier stages. Immutable audit trails should capture all prompts, responses, and system actions. This provides essential evidence for compliance requirements and security incident response. These controls collectively help production systems remain secure, reliable, and compliant while delivering consistent value to users.

Governance and compliance

Generative AI applications in production require governance controls that extend beyond standard software compliance frameworks. The unique characteristics of large language models—

probabilistic systems, emergent capabilities, and potential for unexpected output—create complex risk profiles that require specialized governance approaches. The organization must navigate evolving regulations while establishing robust controls that address ethics, risk, and accountability considerations. This section covers key governance and compliance challenges and controls in production.

Risk management for governance and compliance

Effective governance and compliance for generative AI begins with a comprehensive risk management approach that identifies, evaluates, and mitigates risks. The organization must establish a structured mechanism that addresses known risks and accommodates the discovery of unforeseen challenges.

The following is a risk assessment approach that you can use to address governance and compliance requirements:

1. Risk identification
   - Convene multidisciplinary teams that include legal, compliance, regulatory, security, ethics, business, and technical stakeholders. These teams should validate that the risks are covered adequately.
   - Catalog generative AI-specific risks, including hallucinations, unauthorized disclosure of training data, prompt injection vulnerabilities, and malicious use cases.
   - Conduct stakeholder workshops to identify potential harm across technical, operational, reputational, and legal dimensions.
   - Review the regulatory landscape across deployment jurisdictions to identify compliance obligations. For more information, see Regulatory compliance and controls in this guide.
2. Risk analysis and evaluation
   - Assess the likelihood, frequency, and impact for each identified risk. For more information, see Learn how to assess the risk of AI systems (AWS blog post).
   - Categorize the severity of each risk by using a standardized classification matrix.
   - Prioritize mitigation efforts based on the risk scores and the organizational risk appetite.
   - Document assumptions and limitations in a risk evaluation for future reassessment.
3. Risk treatment and monitoring
   - Maintain an AI risk registry to document and identify AI-specific risks, associated controls, and mitigation mechanisms.

- Implement regular risk review cycles and update procedures to make sure that the risk registry remains current as AI systems evolve and new threats emerge.
- Implement continuous monitoring for key risk indicators, and establish thresholds and escalation pathways.

Regulatory compliance and controls

When it comes to regulatory compliance, organizations must consider requirements based on multiple overlapping regulatory domains. This includes the following:

- **AI-specific regulation** – Organizations need to align with emerging AI-specific regulations, such as the EU AI Act and the NIST AI Risk Management Framework.
- **Data protection laws** – Organizations that capture and use user data and interactions might need to maintain compliance with General Data Protection Regulation (GDPR), the California Consumer Privacy Act (CCPA) and California Privacy Rights Act (CPRA), or other privacy regulations.
- **Industry-specific requirements** – Organizations might also need to address sector-specific regulations, such as the Health Insurance Portability and Accountability Act (HIPAA) for healthcare or the Australian Prudential Regulation Authority (APRA) for finance.
- **Cross-border consideration** – Organizations might also need to navigate data sovereignty requirements when AI systems process information across international borders.

To address these compliance requirements, organizations should implement a comprehensive governance framework that includes the following:

- **Compliance monitoring** – Implement compliance monitoring that tracks regulatory updates and applies relevant policies against the Generative AI application.
- **Standardized documentation** – Have a standardize regulatory documentation process to conform to compliance requirements.
- **Data governance** – Implement metadata tagging for training data to support provenance tracking, verifiable consent management and automated regulatory compliance assessments.
- **Data controls** – Implement and enforce control policies for cross-border data and service utilization to address data sovereignty and regional AI governance frameworks requirements.

- **Regular compliance audits** – Conduct periodic third-party evaluation of AI systems against current regulatory standards to identify potential compliance gaps before they become a regulatory violation.

Ethics and responsible AI

Beyond regulatory compliance and risk management, organizations that deploy generative AI systems need to consider the ethical implications. They must establish operational frameworks for responsible AI that translates principles into practice. Generative AI models create unique ethical challenges around fairness, transparency, and accountability. This requires specialized approaches beyond traditional software development requirements.

Bias detection and mitigation are crucial components of responsible AI implementation. Detection methods include: designing and implementing user feedback channels that specifically capture potential bias issues, deploying automated auditing tools that analyze model output for statistical evidence of bias, and implementing human-in-the-loop mechanisms that sample for bias issues. For mitigation, organizations should establish continuous monitoring systems with feedback loops for model refinement. They should also develop intervention and escalation protocols when bias is detected in production environments.

Human oversight is essential, particularly for high-impact or high-risk decisions. Implementing human-in-the-loop protocols for these scenarios provides an additional layer of scrutiny and decision-making.

A comprehensive, responsible AI framework should be implemented that considers core dimensions across the application. These dimensions include the following:

- **Fairness** – Considering impact on different groups of stakeholders
- **Explainability** – Understanding and evaluating system output
- **Privacy and security** – Appropriately obtaining, using, and protecting data and model
- **Safety** – Preventing harmful system output and misuse
- **Controllability** – Having mechanisms to monitor and steer AI system behavior
- **Veracity and robustness** – Achieving correct system outputs, even with unexpected or adversarial inputs
- **Governance** – Incorporating best practices into the AI supply chain, including providers and deployers

- **Transparency** – Enabling stakeholders to make informed choices about their engagement with an AI system

Robust safeguards that protect data privacy throughout the AI lifecycle are crucial. These include transparent consent mechanisms for data collection, deidentification and anonymization techniques, secure data-storage protocols, privacy-preserving training methods, runtime PII protection, user privacy controls, and continuous privacy monitoring systems.

Organizations should establish an ethics review board to evaluate edge cases and policy decisions. Transparent disclosure practices should be implemented to make sure that users understand when they are interacting with AI systems. Finally, explainability requirements should be addressed across different use cases and stakeholders. This helps you make sure that the AI system's decisions and outputs can be understood and interpreted by relevant parties.

Operational controls

Effective generative AI governance requires operational controls that translate policies into practice. To achieve this, organizations must establish end-to-end oversight mechanisms that provide continuous visibility while facilitating detection and remediation of compliance deviations. Consider the following types of operational controls:

- **Auditability** – Maintain immutable records of the model version, training data, and configuration changes.
- **Logging and tracing** – Establish logging and tracing capabilities that support audits. Maintain application lineage that includes the model, prompt, application version, and data.
- **Monitoring and alerting** – Implement monitoring to provide oversight of the application's performance in production.
- **Incident response** – Develop AI-specific procedures for addressing model failures, security breaches, or harmful outputs.
- **Remediation processes** – Establish processes for addressing identified issues in production models.
- **Compliance mapping** – Define governance requirements and map them to respective compliance requirements and controls.

Scalable maintenance and user support for generative AI applications

Successful production deployment of generative AI applications requires robust maintenance processes and support structures that can scale with increasing usage. This section outlines frameworks for implementing automated maintenance pipelines and establishing comprehensive support systems for both end users and operations teams. It examines strategies for automating routine operational tasks, managing infrastructure at scale, and creating effective support structures through monitoring, ticketing systems, and standardized runbooks. These components form the foundation for maintaining reliable generative AI applications while efficiently supporting a growing user base and increasing system complexity.

Scalable maintenance

The key to scalable maintenance is automation. Repetitive operational tasks should be codified and automated through GenAIOps pipelines. This includes automating the process for retraining or fine-tuning models, upgrading generative AI applications to use the latest models, updating the RAG knowledge base to incorporate new data sources and the latest data, and scaling the underlying cloud infrastructure in response to load. You can achieve this by adopting an infrastructure as code (IaC) and operations as code (OaC) approach. You can use AWS CloudFormation to automate deployment processes, validate consistent environment setup, and streamline resource management. This reduces manual effort, minimizes human error, and validates that the system can be managed efficiently by a small team even as its complexity and usage grow.

User support readiness

As with any production application, the right support structures are essential to make sure that users have a pathway to troubleshoot or escalate. You also need to make sure that business and operations teams have the right tools and mechanisms to respond to an incident.

End user support

The primary goal of end-user support is to help users to effectively use the target application to accomplish their task. This section outlines an example support model that monitors usage and sets up pathways for queries, escalations, and root-cause analysis. You can do the following to establish a robust user support mechanism:

1. Define the end-user support objectives and coverage. Example objectives include support response time, problem resolution time, user downtime, and user feedback. Support issues

typically fall into three categories: guidance and queries, application and business function concerns, and technical issues.

2. Define monitoring metrics that map to these objectives. These metrics serve as key indicators of support effectiveness and help identify areas for improvement.
3. Implement monitoring and alerting systems for two primary purposes: to provide agents with the tools necessary for root-cause analysis, and to enable proactive user engagement when issues arise.
4. Integrate with a ticketing system to streamline support operations. This integration should automate the assignment and routing of issues to relevant agents while providing a mechanism to trace and audit overall support health and track application issues.
5. Design runbooks for root-cause analysis and resolution. These runbooks serve multiple purposes. They facilitate knowledge transfer between agents, enable automated resolution of recurring issues, and provide a systematic mechanism to debug problems and reduce response times.

Business and operations support

Business and operations support focuses on maintaining system reliability to prevent business disruptions. This section outlines the required support model for monitoring application health and development of runbooks/playbooks for diagnosis and resolution. You can do the following to make sure that you are prepared to support the business and its operations:

1. Define business and operational support objectives. Key metrics typically include mean time to recovery (MTTR), system uptime, and recovery time objectives. These fundamental measures help you establish clear performance targets for the support team.
2. Define operational monitoring metrics that map to these objectives. This alignment focuses monitoring efforts on the most critical aspects of system performance.
3. Implement comprehensive monitoring and alerting systems that support engineers with the necessary tools for system diagnostics. These systems should provide real-time visibility into system health and performance.
4. Integrate with a ticketing system to streamline support operations. This automates the assignment and routing of issues to the appropriate engineers, and it provides a mechanism to trace and audit overall support health while tracking operational issues.
5. Design detailed runbooks for root-cause analysis and resolution. These runbooks serve three essential purposes: to facilitate knowledge transfer between engineers, to enable automated

resolution of recurring issues, and to provide a systematic approach to debug errors and reduce response times.

Conclusion

The Generative AI Lifecycle Operational Excellence (GLOE) framework provides a comprehensive and iterative pathway for organizations to navigate the complexities of bringing Generative AI applications from experimental concepts to robust, production-grade systems. The journey through its distinct stages—development, preproduction, and production—transforms generative AI development from an unpredictable art into a mature engineering discipline.

The lifecycle begins in the development (PoC) stage, where the core focus is on rapid, evidence-backed experimentation to de-risk investment by validating business value and technical feasibility.

The lifecycle then matures into the preproduction stage, a pivotal phase dedicated to hardening the application, formalizing a modular and production-intent architecture, and establishing the GenAIOps backbone of CI/CD pipelines, observability, and automated evaluation.

The production stage represents the culmination of this journey. It is the activation of the GLOE framework's core principle: continuous improvement. This is not a static endpoint. It is the beginning of a dynamic, self-sustaining flywheel. The comprehensive monitoring of system health, business impact, and AI quality, combined with robust feedback loops, provides the essential data to fuel this cycle. When monitoring detects drift or user feedback signals a failure, the GLOE framework provides a systematic path to loop back to the appropriate earlier stage.

Ultimately, GLOE helps organizations overcome the gap between prototype and production. By embedding principles of holistic quality assurance, security by design, and automation with human oversight at every step, the framework validates that generative AI solutions are not only innovative but also reliable, secure, and adaptable. This structured, cyclical approach helps enterprises to confidently deploy and manage generative AI applications that deliver consistent performance, adhere to ethical guidelines, and provide sustained, measurable business value throughout their entire lifecycle.

Resources for generative AI development on AWS

Agentic AI

- Operationalizing agentic AI on AWS (AWS Prescriptive Guidance)
- Foundations of agentic AI on AWS (AWS Prescriptive Guidance)
- Agentic AI patterns and workflows on AWS (AWS Prescriptive Guidance)
- Agentic AI frameworks, protocols, and tools on AWS (AWS Prescriptive Guidance)
- Building serverless architectures for agentic AI on AWS (AWS Prescriptive Guidance)
- Building multi-tenant architectures for agentic AI on AWS (AWS Prescriptive Guidance)

AWS AI services

- Amazon Bedrock documentation
- Amazon Comprehend documentation
- Amazon Q Business documentation
- Amazon Q Developer documentation
- Amazon SageMaker AI documentation

ML operations (MLOps)

- Evaluating your ML project with the MLOps checklist (AWS Prescriptive Guidance)
- Planning for successful MLOps (AWS Prescriptive Guidance)

Operational excellence and readiness for generative AI

- Building an enterprise-ready generative AI platform on AWS (AWS Prescriptive Guidance)
- Generative AI workload assessment (AWS Prescriptive Guidance)
- Data security, lifecycle, and strategy for generative AI applications (AWS Prescriptive Guidance)
- Maturity model for adopting generative AI on AWS (AWS Prescriptive Guidance)

Prompt engineering and management

- Prompt engineering best practices to avoid prompt injection attacks on modern LLMs (AWS Prescriptive Guidance)

Retrieval Augmented Generation (RAG)

- Retrieval Augmented Generation options and architectures on AWS (AWS Prescriptive Guidance)

Contributors

Authoring

- Melanie Li, AWS Senior Solutions Architect
- Derrick Choo, AWS Senior Solutions Architect
- Aaron Su, AWS Solutions Architect
- Deepak Dalakoti, AWS Deep Learning Architect
- Mahsa Paknezhad, AWS Deep Learning Architect
- Nicholas Moore, AWS Solutions Architect
- Faisal Masood, AWS Senior Specialist Solutions Architect Containers
- Vincent Wang, AWS Solutions Architect
- Dustin Liu, AWS Solutions Architect
- Rafa Xu, AWS Senior Delivery Consultant
- James Schafer, AWS Security Solutions Architect
- Romain Vivier, AWS Solutions Architect Manager
- Eng-Hwa Tan, AWS Principal Specialist Solutions Architect Containers

Reviewing

- Ramesh Natarajan, AWS Senior Delivery Consultant
- Sovik Nath, AWS Senior Solutions Architect
- Tony Trinh, AWS Senior Solutions Architect
- Alessandro Cerè, AWS Principal Solutions Architect

Technical writing

- Lilly AbouHarb, AWS Senior Technical Writer

Document history

The following table describes significant changes to this guide. If you want to be notified about future updates, you can subscribe to an RSS feed.

Change	Description	Date
Initial publication	—	November 12, 2025

AWS Prescriptive Guidance glossary

The following are commonly used terms in strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

Numbers

7 Rs

Seven common migration strategies for moving applications to the cloud. These strategies build upon the 5 Rs that Gartner identified in 2011 and consist of the following:

- Refactor/re-architect – Move an application and modify its architecture by taking full advantage of cloud-native features to improve agility, performance, and scalability. This typically involves porting the operating system and database. Example: Migrate your on-premises Oracle database to the Amazon Aurora PostgreSQL-Compatible Edition.
- Replatform (lift and reshape) – Move an application to the cloud, and introduce some level of optimization to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Amazon Relational Database Service (Amazon RDS) for Oracle in the AWS Cloud.
- Repurchase (drop and shop) – Switch to a different product, typically by moving from a traditional license to a SaaS model. Example: Migrate your customer relationship management (CRM) system to Salesforce.com.
- Rehost (lift and shift) – Move an application to the cloud without making any changes to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Oracle on an EC2 instance in the AWS Cloud.
- Relocate (hypervisor-level lift and shift) – Move infrastructure to the cloud without purchasing new hardware, rewriting applications, or modifying your existing operations. You migrate servers from an on-premises platform to a cloud service for the same platform. Example: Migrate a Microsoft Hyper-V application to AWS.
- Retain (revisit) – Keep applications in your source environment. These might include applications that require major refactoring, and you want to postpone that work until a later time, and legacy applications that you want to retain, because there's no business justification for migrating them.

- Retire – Decommission or remove applications that are no longer needed in your source environment.

A

ABAC

See attribute-based access control.

abstracted services

See managed services.

ACID

See atomicity, consistency, isolation, durability.

active-active migration

A database migration method in which the source and target databases are kept in sync (by using a bidirectional replication tool or dual write operations), and both databases handle transactions from connecting applications during migration. This method supports migration in small, controlled batches instead of requiring a one-time cutover. It's more flexible but requires more work than active-passive migration.

active-passive migration

A database migration method in which the source and target databases are kept in sync, but only the source database handles transactions from connecting applications while data is replicated to the target database. The target database doesn't accept any transactions during migration.

aggregate function

A SQL function that operates on a group of rows and calculates a single return value for the group. Examples of aggregate functions include `SUM` and `MAX`.

AI

See artificial intelligence.

AIOps

See artificial intelligence operations.

anonymization

The process of permanently deleting personal information in a dataset. Anonymization can help protect personal privacy. Anonymized data is no longer considered to be personal data.

anti-pattern

A frequently used solution for a recurring issue where the solution is counter-productive, ineffective, or less effective than an alternative.

application control

A security approach that allows the use of only approved applications in order to help protect a system from malware.

application portfolio

A collection of detailed information about each application used by an organization, including the cost to build and maintain the application, and its business value. This information is key to the portfolio discovery and analysis process and helps identify and prioritize the applications to be migrated, modernized, and optimized.

artificial intelligence (AI)

The field of computer science that is dedicated to using computing technologies to perform cognitive functions that are typically associated with humans, such as learning, solving problems, and recognizing patterns. For more information, see What is Artificial Intelligence?

artificial intelligence operations (AIOps)

The process of using machine learning techniques to solve operational problems, reduce operational incidents and human intervention, and increase service quality. For more information about how AIOps is used in the AWS migration strategy, see the operations integration guide.

asymmetric encryption

An encryption algorithm that uses a pair of keys, a public key for encryption and a private key for decryption. You can share the public key because it isn't used for decryption, but access to the private key should be highly restricted.

atomicity, consistency, isolation, durability (ACID)

A set of software properties that guarantee the data validity and operational reliability of a database, even in the case of errors, power failures, or other problems.

attribute-based access control (ABAC)

The practice of creating fine-grained permissions based on user attributes, such as department, job role, and team name. For more information, see ABAC for AWS in the AWS Identity and Access Management (IAM) documentation.

authoritative data source

A location where you store the primary version of data, which is considered to be the most reliable source of information. You can copy data from the authoritative data source to other locations for the purposes of processing or modifying the data, such as anonymizing, redacting, or pseudonymizing it.

Availability Zone

A distinct location within an AWS Region that is insulated from failures in other Availability Zones and provides inexpensive, low-latency network connectivity to other Availability Zones in the same Region.

AWS Cloud Adoption Framework (AWS CAF)

A framework of guidelines and best practices from AWS to help organizations develop an efficient and effective plan to move successfully to the cloud. AWS CAF organizes guidance into six focus areas called perspectives: business, people, governance, platform, security, and operations. The business, people, and governance perspectives focus on business skills and processes; the platform, security, and operations perspectives focus on technical skills and processes. For example, the people perspective targets stakeholders who handle human resources (HR), staffing functions, and people management. For this perspective, AWS CAF provides guidance for people development, training, and communications to help ready the organization for successful cloud adoption. For more information, see the AWS CAF website and the AWS CAF whitepaper.

AWS Workload Qualification Framework (AWS WQF)

A tool that evaluates database migration workloads, recommends migration strategies, and provides work estimates. AWS WQF is included with AWS Schema Conversion Tool (AWS SCT). It analyzes database schemas and code objects, application code, dependencies, and performance characteristics, and provides assessment reports.

B

bad bot

A bot that is intended to disrupt or cause harm to individuals or organizations.

BCP

See business continuity planning.

behavior graph

A unified, interactive view of resource behavior and interactions over time. You can use a behavior graph with Amazon Detective to examine failed logon attempts, suspicious API calls, and similar actions. For more information, see Data in a behavior graph in the Detective documentation.

big-endian system

A system that stores the most significant byte first. See also endianness.

binary classification

A process that predicts a binary outcome (one of two possible classes). For example, your ML model might need to predict problems such as “Is this email spam or not spam?" or "Is this product a book or a car?"

bloom filter

A probabilistic, memory-efficient data structure that is used to test whether an element is a member of a set.

blue/green deployment

A deployment strategy where you create two separate but identical environments. You run the current application version in one environment (blue) and the new application version in the other environment (green). This strategy helps you quickly roll back with minimal impact.

bot

A software application that runs automated tasks over the internet and simulates human activity or interaction. Some bots are useful or beneficial, such as web crawlers that index information on the internet. Some other bots, known as *bad bots*, are intended to disrupt or cause harm to individuals or organizations.

botnet

Networks of bots that are infected by malware and are under the control of a single party, known as a *bot herder* or *bot operator*. Botnets are the best-known mechanism to scale bots and their impact.

branch

A contained area of a code repository. The first branch created in a repository is the *main branch*. You can create a new branch from an existing branch, and you can then develop features or fix bugs in the new branch. A branch you create to build a feature is commonly referred to as a *feature branch*. When the feature is ready for release, you merge the feature branch back into the main branch. For more information, see About branches (GitHub documentation).

break-glass access

In exceptional circumstances and through an approved process, a quick means for a user to gain access to an AWS account that they don't typically have permissions to access. For more information, see the Implement break-glass procedures indicator in the AWS Well-Architected guidance.

brownfield strategy

The existing infrastructure in your environment. When adopting a brownfield strategy for a system architecture, you design the architecture around the constraints of the current systems and infrastructure. If you are expanding the existing infrastructure, you might blend brownfield and greenfield strategies.

buffer cache

The memory area where the most frequently accessed data is stored.

business capability

What a business does to generate value (for example, sales, customer service, or marketing). Microservices architectures and development decisions can be driven by business capabilities. For more information, see the Organized around business capabilities section of the Running containerized microservices on AWS whitepaper.

business continuity planning (BCP)

A plan that addresses the potential impact of a disruptive event, such as a large-scale migration, on operations and enables a business to resume operations quickly.

C

CAF

See AWS Cloud Adoption Framework.

canary deployment

The slow and incremental release of a version to end users. When you are confident, you deploy the new version and replace the current version in its entirety.

CCoE

See Cloud Center of Excellence.

CDC

See change data capture.

change data capture (CDC)

The process of tracking changes to a data source, such as a database table, and recording metadata about the change. You can use CDC for various purposes, such as auditing or replicating changes in a target system to maintain synchronization.

chaos engineering

Intentionally introducing failures or disruptive events to test a system's resilience. You can use AWS Fault Injection Service (AWS FIS) to perform experiments that stress your AWS workloads and evaluate their response.

CI/CD

See continuous integration and continuous delivery.

classification

A categorization process that helps generate predictions. ML models for classification problems predict a discrete value. Discrete values are always distinct from one another. For example, a model might need to evaluate whether or not there is a car in an image.

client-side encryption

Encryption of data locally, before the target AWS service receives it.

Cloud Center of Excellence (CCoE)

A multi-disciplinary team that drives cloud adoption efforts across an organization, including developing cloud best practices, mobilizing resources, establishing migration timelines, and leading the organization through large-scale transformations. For more information, see the CCoE posts on the AWS Cloud Enterprise Strategy Blog.

cloud computing

The cloud technology that is typically used for remote data storage and IoT device management. Cloud computing is commonly connected to edge computing technology.

cloud operating model

In an IT organization, the operating model that is used to build, mature, and optimize one or more cloud environments. For more information, see Building your Cloud Operating Model.

cloud stages of adoption

The four phases that organizations typically go through when they migrate to the AWS Cloud:

- Project – Running a few cloud-related projects for proof of concept and learning purposes
- Foundation – Making foundational investments to scale your cloud adoption (e.g., creating a landing zone, defining a CCoE, establishing an operations model)
- Migration – Migrating individual applications
- Re-invention – Optimizing products and services, and innovating in the cloud

These stages were defined by Stephen Orban in the blog post The Journey Toward Cloud-First & the Stages of Adoption on the AWS Cloud Enterprise Strategy blog. For information about how they relate to the AWS migration strategy, see the migration readiness guide.

CMDB

See configuration management database.

code repository

A location where source code and other assets, such as documentation, samples, and scripts, are stored and updated through version control processes. Common cloud repositories include GitHub or Bitbucket Cloud. Each version of the code is called a *branch*. In a microservice structure, each repository is devoted to a single piece of functionality. A single CI/CD pipeline can use multiple repositories.

cold cache

A buffer cache that is empty, not well populated, or contains stale or irrelevant data. This affects performance because the database instance must read from the main memory or disk, which is slower than reading from the buffer cache.

cold data

Data that is rarely accessed and is typically historical. When querying this kind of data, slow queries are typically acceptable. Moving this data to lower-performing and less expensive storage tiers or classes can reduce costs.

computer vision (CV)

A field of AI that uses machine learning to analyze and extract information from visual formats such as digital images and videos. For example, Amazon SageMaker AI provides image processing algorithms for CV.

configuration drift

For a workload, a configuration change from the expected state. It might cause the workload to become noncompliant, and it's typically gradual and unintentional.

configuration management database (CMDB)

A repository that stores and manages information about a database and its IT environment, including both hardware and software components and their configurations. You typically use data from a CMDB in the portfolio discovery and analysis stage of migration.

conformance pack

A collection of AWS Config rules and remediation actions that you can assemble to customize your compliance and security checks. You can deploy a conformance pack as a single entity in an AWS account and Region, or across an organization, by using a YAML template. For more information, see Conformance packs in the AWS Config documentation.

continuous integration and continuous delivery (CI/CD)

The process of automating the source, build, test, staging, and production stages of the software release process. CI/CD is commonly described as a pipeline. CI/CD can help you automate processes, improve productivity, improve code quality, and deliver faster. For more information, see Benefits of continuous delivery. CD can also stand for *continuous deployment*. For more information, see Continuous Delivery vs. Continuous Deployment.

CV

See computer vision.

D

data at rest

Data that is stationary in your network, such as data that is in storage.

data classification

A process for identifying and categorizing the data in your network based on its criticality and sensitivity. It is a critical component of any cybersecurity risk management strategy because it helps you determine the appropriate protection and retention controls for the data. Data classification is a component of the security pillar in the AWS Well-Architected Framework. For more information, see Data classification.

data drift

A meaningful variation between the production data and the data that was used to train an ML model, or a meaningful change in the input data over time. Data drift can reduce the overall quality, accuracy, and fairness in ML model predictions.

data in transit

Data that is actively moving through your network, such as between network resources.

data mesh

An architectural framework that provides distributed, decentralized data ownership with centralized management and governance.

data minimization

The principle of collecting and processing only the data that is strictly necessary. Practicing data minimization in the AWS Cloud can reduce privacy risks, costs, and your analytics carbon footprint.

data perimeter

A set of preventive guardrails in your AWS environment that help make sure that only trusted identities are accessing trusted resources from expected networks. For more information, see Building a data perimeter on AWS.

data preprocessing

To transform raw data into a format that is easily parsed by your ML model. Preprocessing data can mean removing certain columns or rows and addressing missing, inconsistent, or duplicate values.

data provenance

The process of tracking the origin and history of data throughout its lifecycle, such as how the data was generated, transmitted, and stored.

data subject

An individual whose data is being collected and processed.

data warehouse

A data management system that supports business intelligence, such as analytics. Data warehouses commonly contain large amounts of historical data, and they are typically used for queries and analysis.

database definition language (DDL)

Statements or commands for creating or modifying the structure of tables and objects in a database.

database manipulation language (DML)

Statements or commands for modifying (inserting, updating, and deleting) information in a database.

DDL

See database definition language.

deep ensemble

To combine multiple deep learning models for prediction. You can use deep ensembles to obtain a more accurate prediction or for estimating uncertainty in predictions.

deep learning

An ML subfield that uses multiple layers of artificial neural networks to identify mapping between input data and target variables of interest.

defense-in-depth

An information security approach in which a series of security mechanisms and controls are thoughtfully layered throughout a computer network to protect the confidentiality, integrity, and availability of the network and the data within. When you adopt this strategy on AWS, you add multiple controls at different layers of the AWS Organizations structure to help secure resources. For example, a defense-in-depth approach might combine multi-factor authentication, network segmentation, and encryption.

delegated administrator

In AWS Organizations, a compatible service can register an AWS member account to administer the organization's accounts and manage permissions for that service. This account is called the *delegated administrator* for that service. For more information and a list of compatible services, see Services that work with AWS Organizations in the AWS Organizations documentation.

deployment

The process of making an application, new features, or code fixes available in the target environment. Deployment involves implementing changes in a code base and then building and running that code base in the application's environments.

development environment

See environment.

detective control

A security control that is designed to detect, log, and alert after an event has occurred. These controls are a second line of defense, alerting you to security events that bypassed the preventative controls in place. For more information, see Detective controls in *Implementing security controls on AWS*.

development value stream mapping (DVSM)

A process used to identify and prioritize constraints that adversely affect speed and quality in a software development lifecycle. DVSM extends the value stream mapping process originally designed for lean manufacturing practices. It focuses on the steps and teams required to create and move value through the software development process.

digital twin

A virtual representation of a real-world system, such as a building, factory, industrial equipment, or production line. Digital twins support predictive maintenance, remote monitoring, and production optimization.

dimension table

In a star schema, a smaller table that contains data attributes about quantitative data in a fact table. Dimension table attributes are typically text fields or discrete numbers that behave like text. These attributes are commonly used for query constraining, filtering, and result set labeling.

disaster

An event that prevents a workload or system from fulfilling its business objectives in its primary deployed location. These events can be natural disasters, technical failures, or the result of human actions, such as unintentional misconfiguration or a malware attack.

disaster recovery (DR)

The strategy and process you use to minimize downtime and data loss caused by a disaster. For more information, see Disaster Recovery of Workloads on AWS: Recovery in the Cloud in the AWS Well-Architected Framework.

DML

See database manipulation language.

domain-driven design

An approach to developing a complex software system by connecting its components to evolving domains, or core business goals, that each component serves. This concept was introduced by Eric Evans in his book, *Domain-Driven Design: Tackling Complexity in the Heart of Software* (Boston: Addison-Wesley Professional, 2003). For information about how you can use domain-driven design with the strangler fig pattern, see Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.

DR

See disaster recovery.

drift detection

Tracking deviations from a baselined configuration. For example, you can use AWS CloudFormation to detect drift in system resources, or you can use AWS Control Tower to detect changes in your landing zone that might affect compliance with governance requirements.

DVSM

See development value stream mapping.

E

EDA

See exploratory data analysis.

EDI

See electronic data interchange.

edge computing

The technology that increases the computing power for smart devices at the edges of an IoT network. When compared with cloud computing, edge computing can reduce communication latency and improve response time.

electronic data interchange (EDI)

The automated exchange of business documents between organizations. For more information, see What is Electronic Data Interchange.

encryption

A computing process that transforms plaintext data, which is human-readable, into ciphertext.

encryption key

A cryptographic string of randomized bits that is generated by an encryption algorithm. Keys can vary in length, and each key is designed to be unpredictable and unique.

endianness

The order in which bytes are stored in computer memory. Big-endian systems store the most significant byte first. Little-endian systems store the least significant byte first.

endpoint

See service endpoint.

endpoint service

A service that you can host in a virtual private cloud (VPC) to share with other users. You can create an endpoint service with AWS PrivateLink and grant permissions to other AWS accounts or to AWS Identity and Access Management (IAM) principals. These accounts or principals can connect to your endpoint service privately by creating interface VPC endpoints. For more

information, see Create an endpoint service in the Amazon Virtual Private Cloud (Amazon VPC) documentation.

enterprise resource planning (ERP)

A system that automates and manages key business processes (such as accounting, MES, and project management) for an enterprise.

envelope encryption

The process of encrypting an encryption key with another encryption key. For more information, see Envelope encryption in the AWS Key Management Service (AWS KMS) documentation.

environment

An instance of a running application. The following are common types of environments in cloud computing:

- development environment – An instance of a running application that is available only to the core team responsible for maintaining the application. Development environments are used to test changes before promoting them to upper environments. This type of environment is sometimes referred to as a *test environment*.
- lower environments – All development environments for an application, such as those used for initial builds and tests.
- production environment – An instance of a running application that end users can access. In a CI/CD pipeline, the production environment is the last deployment environment.
- upper environments – All environments that can be accessed by users other than the core development team. This can include a production environment, preproduction environments, and environments for user acceptance testing.

epic

In agile methodologies, functional categories that help organize and prioritize your work. Epics provide a high-level description of requirements and implementation tasks. For example, AWS CAF security epics include identity and access management, detective controls, infrastructure security, data protection, and incident response. For more information about epics in the AWS migration strategy, see the program implementation guide.

ERP

See enterprise resource planning.

exploratory data analysis (EDA)

The process of analyzing a dataset to understand its main characteristics. You collect or aggregate data and then perform initial investigations to find patterns, detect anomalies, and check assumptions. EDA is performed by calculating summary statistics and creating data visualizations.

F

fact table

The central table in a star schema. It stores quantitative data about business operations. Typically, a fact table contains two types of columns: those that contain measures and those that contain a foreign key to a dimension table.

fail fast

A philosophy that uses frequent and incremental testing to reduce the development lifecycle. It is a critical part of an agile approach.

fault isolation boundary

In the AWS Cloud, a boundary such as an Availability Zone, AWS Region, control plane, or data plane that limits the effect of a failure and helps improve the resilience of workloads. For more information, see AWS Fault Isolation Boundaries.

feature branch

See branch.

features

The input data that you use to make a prediction. For example, in a manufacturing context, features could be images that are periodically captured from the manufacturing line.

feature importance

How significant a feature is for a model's predictions. This is usually expressed as a numerical score that can be calculated through various techniques, such as Shapley Additive Explanations (SHAP) and integrated gradients. For more information, see Machine learning model interpretability with AWS.

feature transformation

To optimize data for the ML process, including enriching data with additional sources, scaling values, or extracting multiple sets of information from a single data field. This enables the ML model to benefit from the data. For example, if you break down the “2021-05-27 00:15:37” date into “2021”, “May”, “Thu”, and “15”, you can help the learning algorithm learn nuanced patterns associated with different data components.

few-shot prompting

Providing an LLM with a small number of examples that demonstrate the task and desired output before asking it to perform a similar task. This technique is an application of in-context learning, where models learn from examples (*shots*) that are embedded in prompts. Few-shot prompting can be effective for tasks that require specific formatting, reasoning, or domain knowledge. See also zero-shot prompting.

FGAC

See fine-grained access control.

fine-grained access control (FGAC)

The use of multiple conditions to allow or deny an access request.

flash-cut migration

A database migration method that uses continuous data replication through change data capture to migrate data in the shortest time possible, instead of using a phased approach. The objective is to keep downtime to a minimum.

FM

See foundation model.

foundation model (FM)

A large deep-learning neural network that has been training on massive datasets of generalized and unlabeled data. FMs are capable of performing a wide variety of general tasks, such as understanding language, generating text and images, and conversing in natural language. For more information, see What are Foundation Models.

G

generative AI

A subset of AI models that have been trained on large amounts of data and that can use a simple text prompt to create new content and artifacts, such as images, videos, text, and audio. For more information, see What is Generative AI.

geo blocking

See geographic restrictions.

geographic restrictions (geo blocking)

In Amazon CloudFront, an option to prevent users in specific countries from accessing content distributions. You can use an allow list or block list to specify approved and banned countries. For more information, see Restricting the geographic distribution of your content in the CloudFront documentation.

Gitflow workflow

An approach in which lower and upper environments use different branches in a source code repository. The Gitflow workflow is considered legacy, and the trunk-based workflow is the modern, preferred approach.

golden image

A snapshot of a system or software that is used as a template to deploy new instances of that system or software. For example, in manufacturing, a golden image can be used to provision software on multiple devices and helps improve speed, scalability, and productivity in device manufacturing operations.

greenfield strategy

The absence of existing infrastructure in a new environment. When adopting a greenfield strategy for a system architecture, you can select all new technologies without the restriction of compatibility with existing infrastructure, also known as brownfield. If you are expanding the existing infrastructure, you might blend brownfield and greenfield strategies.

guardrail

A high-level rule that helps govern resources, policies, and compliance across organizational units (OUs). *Preventive guardrails* enforce policies to ensure alignment to compliance standards. They are implemented by using service control policies and IAM permissions boundaries.

Detective guardrails detect policy violations and compliance issues, and generate alerts for remediation. They are implemented by using AWS Config, AWS Security Hub CSPM, Amazon GuardDuty, AWS Trusted Advisor, Amazon Inspector, and custom AWS Lambda checks.

H

HA

See high availability.

heterogeneous database migration

Migrating your source database to a target database that uses a different database engine (for example, Oracle to Amazon Aurora). Heterogeneous migration is typically part of a re-architecting effort, and converting the schema can be a complex task. AWS provides AWS SCT that helps with schema conversions.

high availability (HA)

The ability of a workload to operate continuously, without intervention, in the event of challenges or disasters. HA systems are designed to automatically fail over, consistently deliver high-quality performance, and handle different loads and failures with minimal performance impact.

historian modernization

An approach used to modernize and upgrade operational technology (OT) systems to better serve the needs of the manufacturing industry. A *historian* is a type of database that is used to collect and store data from various sources in a factory.

holdout data

A portion of historical, labeled data that is withheld from a dataset that is used to train a machine learning model. You can use holdout data to evaluate the model performance by comparing the model predictions against the holdout data.

homogeneous database migration

Migrating your source database to a target database that shares the same database engine (for example, Microsoft SQL Server to Amazon RDS for SQL Server). Homogeneous migration is typically part of a rehosting or replatforming effort. You can use native database utilities to migrate the schema.

hot data

Data that is frequently accessed, such as real-time data or recent translational data. This data typically requires a high-performance storage tier or class to provide fast query responses.

hotfix

An urgent fix for a critical issue in a production environment. Due to its urgency, a hotfix is usually made outside of the typical DevOps release workflow.

hypercare period

Immediately following cutover, the period of time when a migration team manages and monitors the migrated applications in the cloud in order to address any issues. Typically, this period is 1–4 days in length. At the end of the hypercare period, the migration team typically transfers responsibility for the applications to the cloud operations team.

I

IaC

See infrastructure as code.

identity-based policy

A policy attached to one or more IAM principals that defines their permissions within the AWS Cloud environment.

idle application

An application that has an average CPU and memory usage between 5 and 20 percent over a period of 90 days. In a migration project, it is common to retire these applications or retain them on premises.

IIoT

See industrial Internet of Things.

immutable infrastructure

A model that deploys new infrastructure for production workloads instead of updating, patching, or modifying the existing infrastructure. Immutable infrastructures are inherently more consistent, reliable, and predictable than mutable infrastructure. For more information, see the Deploy using immutable infrastructure best practice in the AWS Well-Architected Framework.

inbound (ingress) VPC

In an AWS multi-account architecture, a VPC that accepts, inspects, and routes network connections from outside an application. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

incremental migration

A cutover strategy in which you migrate your application in small parts instead of performing a single, full cutover. For example, you might move only a few microservices or users to the new system initially. After you verify that everything is working properly, you can incrementally move additional microservices or users until you can decommission your legacy system. This strategy reduces the risks associated with large migrations.

Industry 4.0

A term that was introduced by Klaus Schwab in 2016 to refer to the modernization of manufacturing processes through advances in connectivity, real-time data, automation, analytics, and AI/ML.

infrastructure

All of the resources and assets contained within an application's environment.

infrastructure as code (IaC)

The process of provisioning and managing an application's infrastructure through a set of configuration files. IaC is designed to help you centralize infrastructure management, standardize resources, and scale quickly so that new environments are repeatable, reliable, and consistent.

industrial Internet of Things (IIoT)

The use of internet-connected sensors and devices in the industrial sectors, such as manufacturing, energy, automotive, healthcare, life sciences, and agriculture. For more information, see Building an industrial Internet of Things (IIoT) digital transformation strategy.

inspection VPC

In an AWS multi-account architecture, a centralized VPC that manages inspections of network traffic between VPCs (in the same or different AWS Regions), the internet, and on-premises networks. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

Internet of Things (IoT)

: The network of connected physical objects with embedded sensors or processors that communicate with other devices and systems through the internet or over a local communication network. For more information, see What is IoT?

interpretability

: A characteristic of a machine learning model that describes the degree to which a human can understand how the model's predictions depend on its inputs. For more information, see Machine learning model interpretability with AWS.

IoT

: See Internet of Things.

IT information library (ITIL)

: A set of best practices for delivering IT services and aligning these services with business requirements. ITIL provides the foundation for ITSM.

IT service management (ITSM)

: Activities associated with designing, implementing, managing, and supporting IT services for an organization. For information about integrating cloud operations with ITSM tools, see the operations integration guide.

ITIL

: See IT information library.

ITSM

: See IT service management.

L

label-based access control (LBAC)

: An implementation of mandatory access control (MAC) where the users and the data itself are each explicitly assigned a security label value. The intersection between the user security label and data security label determines which rows and columns can be seen by the user.

landing zone

A landing zone is a well-architected, multi-account AWS environment that is scalable and secure. This is a starting point from which your organizations can quickly launch and deploy workloads and applications with confidence in their security and infrastructure environment. For more information about landing zones, see Setting up a secure and scalable multi-account AWS environment.

large language model (LLM)

A deep learning AI model that is pretrained on a vast amount of data. An LLM can perform multiple tasks, such as answering questions, summarizing documents, translating text into other languages, and completing sentences. For more information, see What are LLMs.

large migration

A migration of 300 or more servers.

LBAC

See label-based access control.

least privilege

The security best practice of granting the minimum permissions required to perform a task. For more information, see Apply least-privilege permissions in the IAM documentation.

lift and shift

See 7 Rs.

little-endian system

A system that stores the least significant byte first. See also endianness.

LLM

See large language model.

lower environments

See environment.

M

machine learning (ML)

A type of artificial intelligence that uses algorithms and techniques for pattern recognition and learning. ML analyzes and learns from recorded data, such as Internet of Things (IoT) data, to generate a statistical model based on patterns. For more information, see Machine Learning.

main branch

See branch.

malware

Software that is designed to compromise computer security or privacy. Malware might disrupt computer systems, leak sensitive information, or gain unauthorized access. Examples of malware include viruses, worms, ransomware, Trojan horses, spyware, and keyloggers.

managed services

AWS services for which AWS operates the infrastructure layer, the operating system, and platforms, and you access the endpoints to store and retrieve data. Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB are examples of managed services. These are also known as *abstracted services*.

manufacturing execution system (MES)

A software system for tracking, monitoring, documenting, and controlling production processes that convert raw materials to finished products on the shop floor.

MAP

See Migration Acceleration Program.

mechanism

A complete process in which you create a tool, drive adoption of the tool, and then inspect the results in order to make adjustments. A mechanism is a cycle that reinforces and improves itself as it operates. For more information, see Building mechanisms in the AWS Well-Architected Framework.

member account

All AWS accounts other than the management account that are part of an organization in AWS Organizations. An account can be a member of only one organization at a time.

MES

See manufacturing execution system.

Message Queuing Telemetry Transport (MQTT)

A lightweight, machine-to-machine (M2M) communication protocol, based on the publish/subscribe pattern, for resource-constrained IoT devices.

microservice

A small, independent service that communicates over well-defined APIs and is typically owned by small, self-contained teams. For example, an insurance system might include microservices that map to business capabilities, such as sales or marketing, or subdomains, such as purchasing, claims, or analytics. The benefits of microservices include agility, flexible scaling, easy deployment, reusable code, and resilience. For more information, see Integrating microservices by using AWS serverless services.

microservices architecture

An approach to building an application with independent components that run each application process as a microservice. These microservices communicate through a well-defined interface by using lightweight APIs. Each microservice in this architecture can be updated, deployed, and scaled to meet demand for specific functions of an application. For more information, see Implementing microservices on AWS.

Migration Acceleration Program (MAP)

An AWS program that provides consulting support, training, and services to help organizations build a strong operational foundation for moving to the cloud, and to help offset the initial cost of migrations. MAP includes a migration methodology for executing legacy migrations in a methodical way and a set of tools to automate and accelerate common migration scenarios.

migration at scale

The process of moving the majority of the application portfolio to the cloud in waves, with more applications moved at a faster rate in each wave. This phase uses the best practices and lessons learned from the earlier phases to implement a *migration factory* of teams, tools, and processes to streamline the migration of workloads through automation and agile delivery. This is the third phase of the AWS migration strategy.

migration factory

Cross-functional teams that streamline the migration of workloads through automated, agile approaches. Migration factory teams typically include operations, business analysts and owners,

migration engineers, developers, and DevOps professionals working in sprints. Between 20 and 50 percent of an enterprise application portfolio consists of repeated patterns that can be optimized by a factory approach. For more information, see the discussion of migration factories and the Cloud Migration Factory guide in this content set.

migration metadata

The information about the application and server that is needed to complete the migration. Each migration pattern requires a different set of migration metadata. Examples of migration metadata include the target subnet, security group, and AWS account.

migration pattern

A repeatable migration task that details the migration strategy, the migration destination, and the migration application or service used. Example: Rehost migration to Amazon EC2 with AWS Application Migration Service.

Migration Portfolio Assessment (MPA)

An online tool that provides information for validating the business case for migrating to the AWS Cloud. MPA provides detailed portfolio assessment (server right-sizing, pricing, TCO comparisons, migration cost analysis) as well as migration planning (application data analysis and data collection, application grouping, migration prioritization, and wave planning). The MPA tool (requires login) is available free of charge to all AWS consultants and APN Partner consultants.

Migration Readiness Assessment (MRA)

The process of gaining insights about an organization's cloud readiness status, identifying strengths and weaknesses, and building an action plan to close identified gaps, using the AWS CAF. For more information, see the migration readiness guide. MRA is the first phase of the AWS migration strategy.

migration strategy

The approach used to migrate a workload to the AWS Cloud. For more information, see the 7 Rs entry in this glossary and see Mobilize your organization to accelerate large-scale migrations.

ML

See machine learning.

modernization

Transforming an outdated (legacy or monolithic) application and its infrastructure into an agile, elastic, and highly available system in the cloud to reduce costs, gain efficiencies, and take advantage of innovations. For more information, see Strategy for modernizing applications in the AWS Cloud.

modernization readiness assessment

An evaluation that helps determine the modernization readiness of an organization's applications; identifies benefits, risks, and dependencies; and determines how well the organization can support the future state of those applications. The outcome of the assessment is a blueprint of the target architecture, a roadmap that details development phases and milestones for the modernization process, and an action plan for addressing identified gaps. For more information, see Evaluating modernization readiness for applications in the AWS Cloud.

monolithic applications (monoliths)

Applications that run as a single service with tightly coupled processes. Monolithic applications have several drawbacks. If one application feature experiences a spike in demand, the entire architecture must be scaled. Adding or improving a monolithic application's features also becomes more complex when the code base grows. To address these issues, you can use a microservices architecture. For more information, see Decomposing monoliths into microservices.

MPA

See Migration Portfolio Assessment.

MQTT

See Message Queuing Telemetry Transport.

multiclass classification

A process that helps generate predictions for multiple classes (predicting one of more than two outcomes). For example, an ML model might ask "Is this product a book, car, or phone?" or "Which product category is most interesting to this customer?"

mutable infrastructure

A model that updates and modifies the existing infrastructure for production workloads. For improved consistency, reliability, and predictability, the AWS Well-Architected Framework recommends the use of immutable infrastructure as a best practice.

O

OAC

See origin access control.

OAI

See origin access identity.

OCM

See organizational change management.

offline migration

A migration method in which the source workload is taken down during the migration process. This method involves extended downtime and is typically used for small, non-critical workloads.

OI

See operations integration.

OLA

See operational-level agreement.

online migration

A migration method in which the source workload is copied to the target system without being taken offline. Applications that are connected to the workload can continue to function during the migration. This method involves zero to minimal downtime and is typically used for critical production workloads.

OPC-UA

See Open Process Communications - Unified Architecture.

Open Process Communications - Unified Architecture (OPC-UA)

A machine-to-machine (M2M) communication protocol for industrial automation. OPC-UA provides an interoperability standard with data encryption, authentication, and authorization schemes.

operational-level agreement (OLA)

An agreement that clarifies what functional IT groups promise to deliver to each other, to support a service-level agreement (SLA).

operational readiness review (ORR)

A checklist of questions and associated best practices that help you understand, evaluate, prevent, or reduce the scope of incidents and possible failures. For more information, see Operational Readiness Reviews (ORR) in the AWS Well-Architected Framework.

operational technology (OT)

Hardware and software systems that work with the physical environment to control industrial operations, equipment, and infrastructure. In manufacturing, the integration of OT and information technology (IT) systems is a key focus for Industry 4.0 transformations.

operations integration (OI)

The process of modernizing operations in the cloud, which involves readiness planning, automation, and integration. For more information, see the operations integration guide.

organization trail

A trail that's created by AWS CloudTrail that logs all events for all AWS accounts in an organization in AWS Organizations. This trail is created in each AWS account that's part of the organization and tracks the activity in each account. For more information, see Creating a trail for an organization in the CloudTrail documentation.

organizational change management (OCM)

A framework for managing major, disruptive business transformations from a people, culture, and leadership perspective. OCM helps organizations prepare for, and transition to, new systems and strategies by accelerating change adoption, addressing transitional issues, and driving cultural and organizational changes. In the AWS migration strategy, this framework is called *people acceleration*, because of the speed of change required in cloud adoption projects. For more information, see the OCM guide.

origin access control (OAC)

In CloudFront, an enhanced option for restricting access to secure your Amazon Simple Storage Service (Amazon S3) content. OAC supports all S3 buckets in all AWS Regions, server-side encryption with AWS KMS (SSE-KMS), and dynamic PUT and DELETE requests to the S3 bucket.

origin access identity (OAI)

In CloudFront, an option for restricting access to secure your Amazon S3 content. When you use OAI, CloudFront creates a principal that Amazon S3 can authenticate with. Authenticated principals can access content in an S3 bucket only through a specific CloudFront distribution. See also OAC, which provides more granular and enhanced access control.

ORR

See operational readiness review.

OT

See operational technology.

outbound (egress) VPC

In an AWS multi-account architecture, a VPC that handles network connections that are initiated from within an application. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

P

permissions boundary

An IAM management policy that is attached to IAM principals to set the maximum permissions that the user or role can have. For more information, see Permissions boundaries in the IAM documentation.

personally identifiable information (PII)

Information that, when viewed directly or paired with other related data, can be used to reasonably infer the identity of an individual. Examples of PII include names, addresses, and contact information.

PII

See personally identifiable information.

playbook

A set of predefined steps that capture the work associated with migrations, such as delivering core operations functions in the cloud. A playbook can take the form of scripts, automated runbooks, or a summary of processes or steps required to operate your modernized environment.

PLC

See programmable logic controller.

PLM

See product lifecycle management.

policy

An object that can define permissions (see identity-based policy), specify access conditions (see resource-based policy), or define the maximum permissions for all accounts in an organization in AWS Organizations (see service control policy).

polyglot persistence

Independently choosing a microservice's data storage technology based on data access patterns and other requirements. If your microservices have the same data storage technology, they can encounter implementation challenges or experience poor performance. Microservices are more easily implemented and achieve better performance and scalability if they use the data store best adapted to their requirements.

portfolio assessment

A process of discovering, analyzing, and prioritizing the application portfolio in order to plan the migration. For more information, see Evaluating migration readiness.

predicate

A query condition that returns `true` or `false`, commonly located in a `WHERE` clause.

predicate pushdown

A database query optimization technique that filters the data in the query before transfer. This reduces the amount of data that must be retrieved and processed from the relational database, and it improves query performance.

preventative control

A security control that is designed to prevent an event from occurring. These controls are a first line of defense to help prevent unauthorized access or unwanted changes to your network. For more information, see Preventative controls in *Implementing security controls on AWS*.

principal

An entity in AWS that can perform actions and access resources. This entity is typically a root user for an AWS account, an IAM role, or a user. For more information, see *Principal* in Roles terms and concepts in the IAM documentation.

privacy by design

A system engineering approach that takes privacy into account through the whole development process.

private hosted zones

A container that holds information about how you want Amazon Route 53 to respond to DNS queries for a domain and its subdomains within one or more VPCs. For more information, see Working with private hosted zones in the Route 53 documentation.

proactive control

A security control designed to prevent the deployment of noncompliant resources. These controls scan resources before they are provisioned. If the resource is not compliant with the control, then it isn't provisioned. For more information, see the Controls reference guide in the AWS Control Tower documentation and see Proactive controls in *Implementing security controls on AWS*.

product lifecycle management (PLM)

The management of data and processes for a product throughout its entire lifecycle, from design, development, and launch, through growth and maturity, to decline and removal.

production environment

See environment.

programmable logic controller (PLC)

In manufacturing, a highly reliable, adaptable computer that monitors machines and automates manufacturing processes.

prompt chaining

Using the output of one LLM prompt as the input for the next prompt to generate better responses. This technique is used to break down a complex task into subtasks, or to iteratively refine or expand a preliminary response. It helps improve the accuracy and relevance of a model's responses and allows for more granular, personalized results.

pseudonymization

The process of replacing personal identifiers in a dataset with placeholder values. Pseudonymization can help protect personal privacy. Pseudonymized data is still considered to be personal data.

publish/subscribe (pub/sub)

A pattern that enables asynchronous communications among microservices to improve scalability and responsiveness. For example, in a microservices-based MES, a microservice can publish event messages to a channel that other microservices can subscribe to. The system can add new microservices without changing the publishing service.

Q

query plan

A series of steps, like instructions, that are used to access the data in a SQL relational database system.

query plan regression

When a database service optimizer chooses a less optimal plan than it did before a given change to the database environment. This can be caused by changes to statistics, constraints, environment settings, query parameter bindings, and updates to the database engine.

R

RACI matrix

See responsible, accountable, consulted, informed (RACI).

RAG

See Retrieval Augmented Generation.

ransomware

A malicious software that is designed to block access to a computer system or data until a payment is made.

RASCI matrix

See responsible, accountable, consulted, informed (RACI).

RCAC

See row and column access control.

read replica

A copy of a database that's used for read-only purposes. You can route queries to the read replica to reduce the load on your primary database.

re-architect

See 7 Rs.

recovery point objective (RPO)

The maximum acceptable amount of time since the last data recovery point. This determines what is considered an acceptable loss of data between the last recovery point and the interruption of service.

recovery time objective (RTO)

The maximum acceptable delay between the interruption of service and restoration of service.

refactor

See 7 Rs.

Region

A collection of AWS resources in a geographic area. Each AWS Region is isolated and independent of the others to provide fault tolerance, stability, and resilience. For more information, see Specify which AWS Regions your account can use.

regression

An ML technique that predicts a numeric value. For example, to solve the problem of "What price will this house sell for?" an ML model could use a linear regression model to predict a house's sale price based on known facts about the house (for example, the square footage).

rehost

See 7 Rs.

release

In a deployment process, the act of promoting changes to a production environment.

relocate

See 7 Rs.

replatform

See 7 Rs.

repurchase

See 7 Rs.

resiliency

An application's ability to resist or recover from disruptions. High availability and disaster recovery are common considerations when planning for resiliency in the AWS Cloud. For more information, see AWS Cloud Resilience.

resource-based policy

A policy attached to a resource, such as an Amazon S3 bucket, an endpoint, or an encryption key. This type of policy specifies which principals are allowed access, supported actions, and any other conditions that must be met.

responsible, accountable, consulted, informed (RACI) matrix

A matrix that defines the roles and responsibilities for all parties involved in migration activities and cloud operations. The matrix name is derived from the responsibility types defined in the matrix: responsible (R), accountable (A), consulted (C), and informed (I). The support (S) type is optional. If you include support, the matrix is called a *RASCI matrix*, and if you exclude it, it's called a *RACI matrix*.

responsive control

A security control that is designed to drive remediation of adverse events or deviations from your security baseline. For more information, see Responsive controls in *Implementing security controls on AWS*.

retain

See 7 Rs.

retire

See 7 Rs.

Retrieval Augmented Generation (RAG)

A generative AI technology in which an LLM references an authoritative data source that is outside of its training data sources before generating a response. For example, a RAG model might perform a semantic search of an organization's knowledge base or custom data. For more information, see What is RAG.

rotation

The process of periodically updating a secret to make it more difficult for an attacker to access the credentials.

row and column access control (RCAC)

The use of basic, flexible SQL expressions that have defined access rules. RCAC consists of row permissions and column masks.

RPO

See recovery point objective.

RTO

See recovery time objective.

runbook

A set of manual or automated procedures required to perform a specific task. These are typically built to streamline repetitive operations or procedures with high error rates.

S

SAML 2.0

An open standard that many identity providers (IdPs) use. This feature enables federated single sign-on (SSO), so users can log into the AWS Management Console or call the AWS API operations without you having to create user in IAM for everyone in your organization. For more information about SAML 2.0-based federation, see About SAML 2.0-based federation in the IAM documentation.

SCADA

See supervisory control and data acquisition.

SCP

See service control policy.

secret

In AWS Secrets Manager, confidential or restricted information, such as a password or user credentials, that you store in encrypted form. It consists of the secret value and its metadata.

The secret value can be binary, a single string, or multiple strings. For more information, see What's in a Secrets Manager secret? in the Secrets Manager documentation.

security by design

A system engineering approach that takes security into account through the whole development process.

security control

A technical or administrative guardrail that prevents, detects, or reduces the ability of a threat actor to exploit a security vulnerability. There are four primary types of security controls: preventative, detective, responsive, and proactive.

security hardening

The process of reducing the attack surface to make it more resistant to attacks. This can include actions such as removing resources that are no longer needed, implementing the security best practice of granting least privilege, or deactivating unnecessary features in configuration files.

security information and event management (SIEM) system

Tools and services that combine security information management (SIM) and security event management (SEM) systems. A SIEM system collects, monitors, and analyzes data from servers, networks, devices, and other sources to detect threats and security breaches, and to generate alerts.

security response automation

A predefined and programmed action that is designed to automatically respond to or remediate a security event. These automations serve as detective or responsive security controls that help you implement AWS security best practices. Examples of automated response actions include modifying a VPC security group, patching an Amazon EC2 instance, or rotating credentials.

server-side encryption

Encryption of data at its destination, by the AWS service that receives it.

service control policy (SCP)

A policy that provides centralized control over permissions for all accounts in an organization in AWS Organizations. SCPs define guardrails or set limits on actions that an administrator can delegate to users or roles. You can use SCPs as allow lists or deny lists, to specify which services or actions are permitted or prohibited. For more information, see Service control policies in the AWS Organizations documentation.

service endpoint

The URL of the entry point for an AWS service. You can use the endpoint to connect programmatically to the target service. For more information, see AWS service endpoints in *AWS General Reference*.

service-level agreement (SLA)

An agreement that clarifies what an IT team promises to deliver to their customers, such as service uptime and performance.

service-level indicator (SLI)

A measurement of a performance aspect of a service, such as its error rate, availability, or throughput.

service-level objective (SLO)

A target metric that represents the health of a service, as measured by a service-level indicator.

shared responsibility model

A model describing the responsibility you share with AWS for cloud security and compliance. AWS is responsible for security *of* the cloud, whereas you are responsible for security *in* the cloud. For more information, see Shared responsibility model.

SIEM

See security information and event management system.

single point of failure (SPOF)

A failure in a single, critical component of an application that can disrupt the system.

SLA

See service-level agreement.

SLI

See service-level indicator.

SLO

See service-level objective.

split-and-seed model

A pattern for scaling and accelerating modernization projects. As new features and product releases are defined, the core team splits up to create new product teams. This helps scale your

organization's capabilities and services, improves developer productivity, and supports rapid innovation. For more information, see Phased approach to modernizing applications in the AWS Cloud.

SPOF

See single point of failure.

star schema

A database organizational structure that uses one large fact table to store transactional or measured data and uses one or more smaller dimensional tables to store data attributes. This structure is designed for use in a data warehouse or for business intelligence purposes.

strangler fig pattern

An approach to modernizing monolithic systems by incrementally rewriting and replacing system functionality until the legacy system can be decommissioned. This pattern uses the analogy of a fig vine that grows into an established tree and eventually overcomes and replaces its host. The pattern was introduced by Martin Fowler as a way to manage risk when rewriting monolithic systems. For an example of how to apply this pattern, see Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.

subnet

A range of IP addresses in your VPC. A subnet must reside in a single Availability Zone.

supervisory control and data acquisition (SCADA)

In manufacturing, a system that uses hardware and software to monitor physical assets and production operations.

symmetric encryption

An encryption algorithm that uses the same key to encrypt and decrypt the data.

synthetic testing

Testing a system in a way that simulates user interactions to detect potential issues or to monitor performance. You can use Amazon CloudWatch Synthetics to create these tests.

system prompt

A technique for providing context, instructions, or guidelines to an LLM to direct its behavior. System prompts help set context and establish rules for interactions with users.

T

tags

Key-value pairs that act as metadata for organizing your AWS resources. Tags can help you manage, identify, organize, search for, and filter resources. For more information, see Tagging your AWS resources.

target variable

The value that you are trying to predict in supervised ML. This is also referred to as an *outcome variable*. For example, in a manufacturing setting the target variable could be a product defect.

task list

A tool that is used to track progress through a runbook. A task list contains an overview of the runbook and a list of general tasks to be completed. For each general task, it includes the estimated amount of time required, the owner, and the progress.

test environment

See environment.

training

To provide data for your ML model to learn from. The training data must contain the correct answer. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that you want to predict). It outputs an ML model that captures these patterns. You can then use the ML model to make predictions on new data for which you don't know the target.

transit gateway

A network transit hub that you can use to interconnect your VPCs and on-premises networks. For more information, see What is a transit gateway in the AWS Transit Gateway documentation.

trunk-based workflow

An approach in which developers build and test features locally in a feature branch and then merge those changes into the main branch. The main branch is then built to the development, preproduction, and production environments, sequentially.

trusted access

Granting permissions to a service that you specify to perform tasks in your organization in AWS Organizations and in its accounts on your behalf. The trusted service creates a service-linked role in each account, when that role is needed, to perform management tasks for you. For more information, see Using AWS Organizations with other AWS services in the AWS Organizations documentation.

tuning

To change aspects of your training process to improve the ML model's accuracy. For example, you can train the ML model by generating a labeling set, adding labels, and then repeating these steps several times under different settings to optimize the model.

two-pizza team

A small DevOps team that you can feed with two pizzas. A two-pizza team size ensures the best possible opportunity for collaboration in software development.

U

uncertainty

A concept that refers to imprecise, incomplete, or unknown information that can undermine the reliability of predictive ML models. There are two types of uncertainty: *Epistemic uncertainty* is caused by limited, incomplete data, whereas *aleatoric uncertainty* is caused by the noise and randomness inherent in the data.

undifferentiated tasks

Also known as *heavy lifting*, work that is necessary to create and operate an application but that doesn't provide direct value to the end user or provide competitive advantage. Examples of undifferentiated tasks include procurement, maintenance, and capacity planning.

upper environments

See environment.

V

vacuuming

A database maintenance operation that involves cleaning up after incremental updates to reclaim storage and improve performance.

version control

Processes and tools that track changes, such as changes to source code in a repository.

VPC peering

A connection between two VPCs that allows you to route traffic by using private IP addresses. For more information, see What is VPC peering in the Amazon VPC documentation.

vulnerability

A software or hardware flaw that compromises the security of the system.

W

warm cache

A buffer cache that contains current, relevant data that is frequently accessed. The database instance can read from the buffer cache, which is faster than reading from the main memory or disk.

warm data

Data that is infrequently accessed. When querying this kind of data, moderately slow queries are typically acceptable.

window function

A SQL function that performs a calculation on a group of rows that relate in some way to the current record. Window functions are useful for processing tasks, such as calculating a moving average or accessing the value of rows based on the relative position of the current row.

workload

A collection of resources and code that delivers business value, such as a customer-facing application or backend process.

workstream

Functional groups in a migration project that are responsible for a specific set of tasks. Each workstream is independent but supports the other workstreams in the project. For example, the portfolio workstream is responsible for prioritizing applications, wave planning, and collecting migration metadata. The portfolio workstream delivers these assets to the migration workstream, which then migrates the servers and applications.

WORM

See write once, read many.

WQF

See AWS Workload Qualification Framework.

write once, read many (WORM)

A storage model that writes data a single time and prevents the data from being deleted or modified. Authorized users can read the data as many times as needed, but they cannot change it. This data storage infrastructure is considered immutable.

Z

zero-day exploit

An attack, typically malware, that takes advantage of a zero-day vulnerability.

zero-day vulnerability

An unmitigated flaw or vulnerability in a production system. Threat actors can use this type of vulnerability to attack the system. Developers frequently become aware of the vulnerability as a result of the attack.

zero-shot prompting

Providing an LLM with instructions for performing a task but no examples (*shots*) that can help guide it. The LLM must use its pre-trained knowledge to handle the task. The effectiveness of zero-shot prompting depends on the complexity of the task and the quality of the prompt. See also few-shot prompting.

zombie application

An application that has an average CPU and memory usage below 5 percent. In a migration project, it is common to retire these applications.