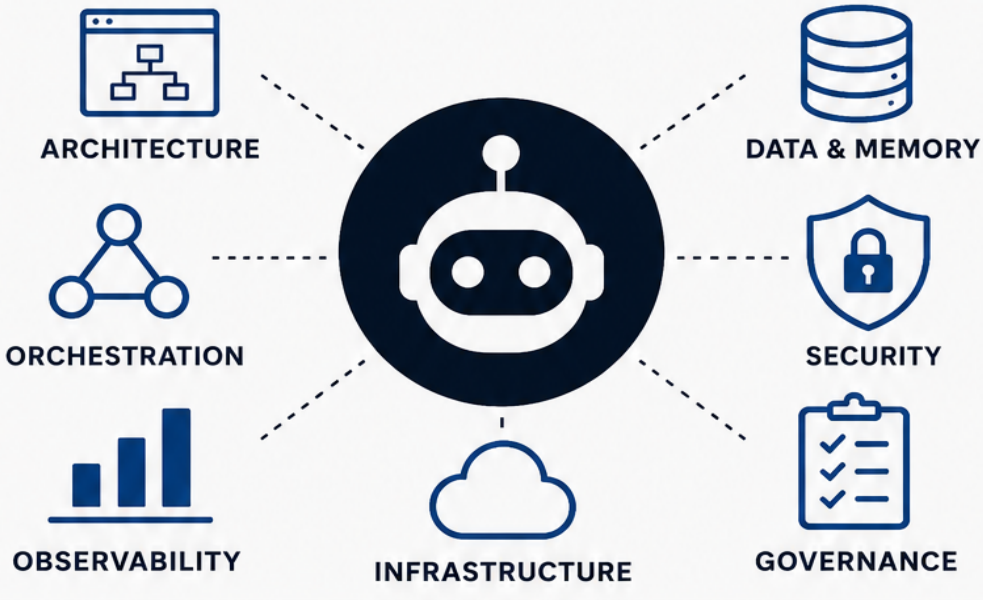
AI AGENTS AND SYSTEMS ENGINEERING

ARCHITECTURE, DESIGN PATTERNS, AND PRODUCTION PRACTICES FOR AUTONOMOUS SYSTEMS



ANDREW GRANT

AI Agents and Systems Engineering

Architecture, Design Patterns, and Production Practices for Autonomous Systems

Steve Publications

This book is available at
https://leanpub.com/aiagentsandsystemsengineering

This version was published on 2026-09-05

Contents

Architecture, Design Patterns, and Production Practices for Autonomous Systems **1**

Introduction **2**
What You Will Learn 2
The Running Example 3
How to Use This Book 4
A Note on Technology and Timeliness 4

Chapter 1: What Are AI Agents and What Problem Do They Solve **6**
Definitions and Terminology: Agents, Tools, Autonomy, and Orchestration 6
The Agent Capability Spectrum: From Stateless Calls to Autonomous Loops 7
Agents vs Chatbots vs Pipelines vs Microservices: Where Each Fits . . 9
The Engineering Problem: Why Agents Are Harder Than Traditional Systems 10
Real-World Problem Spaces: What Agents Are Actually Good For . . . 12
A Running Example: The DesignOps Agent 14
Summary 15

Chapter 2: Historical Foundations from Symbolic AI to LLM Agents . . **16**
Classical Agents: BDI Architectures, SOAR, and Symbolic Planning . . 16
The Rise of Web Agents and AutoAgents (1990s–2000s) 17
Early Autonomous Systems: ROS, Swarm Intelligence, and Multi-Agent Systems 18
Deep Learning and the End of the Symbolic Era 19
GPT-3 and the Birth of Modern LLM Agents 19
Summary 21

Chapter 3: Core Architectural Components of Modern Agents **22**

The Cognitive Stack: Perception, Reasoning, Action, and Reflection . . 22
LLMs as Inference Engines: Capabilities, Limits, and Integration Patterns 22
Tools and Functions: The Agent's Interface to the Real World 22
Memory and State: Short-Term Context vs Long-Term Knowledge . . 22
Planning and Decomposition: From Single-Turn to Multi-Step Reasoning 22
The Minimal Viable Agent Architecture 23
Summary 23

Chapter 4: Reasoning Patterns Prompt Architectures and Cognitive Workflows 24
System Prompts and Role Framing: Setting Behavior Bounds 24
Chain of Thought and Structured Reasoning: Design and Trade-offs . 24
Task Decomposition: Hierarchical and Flat Planning Approaches . . . 24
Reflection and Self-Correction: Critique Loops and Verification 24
Determinism Engineering: Constrained Generation and Output Formatting 24
Failure Modes in Reasoning: Hallucination, Drift, and Mode Collapse . 25
Summary 25

Chapter 5: Tool Use, Function Calling, and External Integration 26
Tool Definition Standards: JSON Schema, OpenAPI, and MCP Schemas 26
Function Calling APIs: How Modern LLMs Bind Tools to Generation . 26
Tool Discovery and Selection: When the Agent Chooses What to Call 26
Error Handling and Recovery: Tool Failures, Timeouts, and Retries . . 26
Security and Sandboxing: Running Agent Tools Safely 26
Building Your Own Tool Server: From Simple APIs to Complex Services 27
Summary 27

Chapter 6: Memory Systems Context Management and Knowledge Retrieval 28
Conversation Memory and Context Windows: The Naive Approach and Its Limits 28
Context Engineering: Summarization, Sliding Windows, and Key-Fact Extraction 28
Vector Stores and Semantic Search: Architecture and Trade-offs . . . 28
Retrieval-Augmented Generation as a System: Not Just a Prompt Trick 28

Hybrid Memory: Combining Episodic, Semantic, and Procedural Storage 28
State Persistence and Recovery: Making Memory Survive Crashes and Restarts 29
Summary 29

Chapter 7: Multi-Agent Systems and Orchestration Patterns 30
Why Multiple Agents: Specialization, Scale, and Separation of Concerns 30
Coordinator-Worker and Manager-Worker Patterns 30
Peer-to-Peer Agent Communication and Consensus 30
Debate and Critique Architectures: Agents That Challenge Each Other 30
Hierarchical Agent Organizations: Hierarchical Task Networks Reborn 30
Orchestration Frameworks: LangGraph, CrewAI, AutoGen, and Alternatives 31
Summary 31

Chapter 8: Workflow Engines and Deterministic Boundaries 32
The Determinism Problem: Why Pure Agent Loops Fail in Production 32
Workflow as the Execution Backbone: Temporal, Dagster, and Friends 32
State Machines and Directed Acyclic Graphs for Agent Workflows . . 32
Idempotency, Retries, and Compensation in Agent Workflows 32
Human Approval Gates and Escalation Paths 32
Mixing Deterministic and Non-Deterministic: The Hybrid Architecture 33
Summary 33

Chapter 9: Agent Runtime Architectures Process Container and Cluster 34
Agent as a Process: Long-Running Services vs Ephemeral Workers . . 34
Containerizing Agent Systems: Dockerfiles, Dependencies, and Base Images 34
Kubernetes for Agent Workloads: Deployments, Jobs, and Custom Resources 34
Serverless and FaaS for Agents: When It Works, When It Breaks 34
Local and Private AI Infrastructure: Running Agents Without Cloud Providers 34
Summary 35

Chapter 10: Model Serving and Inference Infrastructure 36
Model Serving Options: Cloud APIs, Open-Source Models, and Fine-Tuned Models 36

Inference Optimization: Quantization, Speculative Decoding, and Batch Processing 36
Caching Strategies: Prompt/Response Caching, Semantic Caching, and Tool Result Caching 36
Multi-Model Routing: Choosing the Right Model for the Right Task . . 36
Cost Architecture: Token Budgeting, Rate Limiting, and Predictive Scaling 36
Local Model Serving: Ollama, vLLM, TGI, and Self-Hosted Stacks . . . 37
Summary 37

Chapter 11: Asynchronous Execution Queues and Event-Driven Architectures 38
Why Agents Need Async: Latency, Concurrency, and Long-Running Operations 38
Message Queues for Agent Communication: RabbitMQ, Kafka, NATS . 38
Event Sourcing and CQRS with Agents 38
Scheduling and Cron-like Behavior: When Agents Should Run 38
Webhooks and Callbacks: Triggering Agents from External Systems . 38
Handling Timeouts, Dead Letters, and Orphaned Tasks 39
Summary 39

Chapter 12: Communication Protocols and Interoperability 40
API Integration Patterns: REST, gRPC, GraphQL, and Async Patterns . 40
The Model Context Protocol: Specification, Design, and Adoption . . . 40
Agent-to-Agent Communication: Messages, Contracts, and Protocols 40
Event Bus and Pub/Sub for Agent Ecosystems 40
Standardizing Agent Contracts: Schema Validation and Versioning . . 40
Integrating Legacy Systems and Internal Services 41
Summary 41

Chapter 13: Reliability Engineering for Autonomous Agents 42
Defining Reliability for Non-Deterministic Systems 42
Circuit Breakers, Bulkheads, and Timeouts for Agent Operations . . . 42
State Recovery: Checkpointing, Snapshots, and Rollback Strategies . . 42
Graceful Degradation: What Happens When the Model Is Down or Wrong 42
Resilience Patterns: Fallback Models, Degraded Modes, and Human Override 42
Chaos Engineering for Agents: Breaking Things on Purpose 43

Summary 43

Chapter 14: Observability Logging Tracing Metrics and Dashboards . . 44
The Observability Challenge: Non-Determinism Makes Debugging Harder 44
Structured Logging for Agent Systems: Events, Decisions, and Tool Calls 44
Distributed Tracing: End-to-End Visibility Across Agent Chains 44
Metrics That Matter: Latency, Token Usage, Success Rates, and Cost . 44
Dashboards and Alerting: What Engineers Actually Need to See 44
Compliance Logging: Audit Trails for Regulated Environments 45
Summary 45

Chapter 15: Evaluation and Quality Engineering 46
Testing Agents: Unit Tests, Integration Tests, and End-to-End Scenarios 46
Evaluation Frameworks: LLM-as-Judge, Rule-Based Checks, and Human Review 46
Adversarial Testing: Prompt Injection, Jailbreak Attempts, and Attack Simulation 46
Regression Testing for Non-Deterministic Systems 46
Golden Datasets and Continuous Evaluation in Production 46
Cost and Performance Benchmarks: Measuring Efficiency, Not Just Accuracy 47
Summary 47

Chapter 16: Security Architecture for Agent Systems 48
The Expanded Attack Surface: What Makes Agents Harder to Secure . 48
Prompt Injection and Indirect Prompt Injection: Vectors and Defenses 48
Tool Abuse and Escalation: Preventing Agents from Doing Harm 48
Sandboxing and Isolation: Containers, Wasm, and Restricted Runtimes 48
Least-Privilege Execution: Identity, Roles, and Permission Models . . 48
Secrets Management and Credential Handling in Agent Systems 49
Summary 49

Chapter 17: Data Security Privacy and Compliance 50
Data Classification and Handling: What Agents See and Where It Goes 50
PII, PHI, and Regulated Data: Requirements and Architectural Responses 50
Encryption: At Rest, In Transit, and In Use 50
Data Minimization and Retention Policies for Agent Memory 50
Audit Trails and Provenance: Tracing What Data Was Used for What Decision 50

Regulatory Landscape: EU AI Act, SOC2, HIPAA, GDPR, and Industry-Specific Rules . . . 51
Summary . . . 51

Chapter 18: Governance Guardrails and Responsible Operation . . . 52
The Governance Problem: Who Is Responsible When the Agent Acts? 52
Policy Enforcement: Guardrails, Constraints, and Allow/Deny Lists . . 52
Human-in-the-Loop Design: Approval, Override, and Escalation Patterns . . . 52
Model Governance: Versioning, Testing, and Deployment of Foundation Models . . . 52
Incident Response: What to Do When an Agent Goes Rogue . . . 52
Organizational Practices: Teams, Processes, and Decision Rights . . . 53
Summary . . . 53

Chapter 19: Scalability and Performance Optimization . . . 54
Scaling Patterns: Horizontal Scaling, Sharding, and Load Balancing for Agents . . . 54
Concurrency Models: Handling Many Agents and Many Tools Simultaneously . . . 54
Performance Tuning: Reducing Latency, Token Overhead, and Tool Call Chains . . . 54
Rate Limiting and Backpressure: Protecting Models and Tools from Overload . . . 54
Capacity Planning: Sizing Infrastructure for Agent Workloads . . . 54
Cost Optimization: Architectural Choices That Reduce Expense . . . 55
Summary . . . 55

Chapter 20: CI/CD Infrastructure as Code and DevOps for Agent Systems 56
Versioning Agent Code, Prompts, and Configurations . . . 56
CI/CD Pipelines for Agent Systems: Testing, Building, and Deploying 56
Infrastructure as Code: Terraform, Pulumi, and Kubernetes Manifests 56
Configuration Management: Environment-Specific Config and Feature Flags . . . 56
Blue-Green and Canary Deployments for Agent Systems . . . 56
Rollback Strategies: Reverting Agents and Their State . . . 57
Summary . . . 57

Chapter 21: Reference Architectures From Prototype to Enterprise . . . 58
Architecture Level 1: Single Agent, Single Model, Simple Tools . . . 58

Architecture Level 2: Multi-Tool Agent with State and Memory 58
Architecture Level 3: Orchestrated Multi-Agent System with Workflow Engine 58
Architecture Level 4: Enterprise Platform with Security, Observability, and Governance 58
Evolution Paths: How to Grow Without Rewriting 58
Technology Stack Recommendations by Use Case and Scale 59
Summary 59

Chapter 22: Conclusion The Future of Agent-Centric Software Architecture 60
Lessons Learned: Architectural Principles for Longevity 60
What We Still Do Not Know: Open Research and Engineering Problems 60
How Agents Change Software Architecture: A Structural Shift 60
Agents in Developer Tooling: Self-Improving Systems and AI-Assisted Engineering 60
Agents in IT Operations: Autonomous SRE and Self-Healing Infrastructure 60
Final Thoughts: Engineering Discipline in an Age of Autonomy 61

References 62

Architecture, Design Patterns, and Production Practices for Autonomous Systems

This book is a technical guide to designing, building, and operating production-grade AI agent systems. It treats agents as a class of distributed, non-deterministic software that demands architectural discipline, reliability engineering, and security practices adapted for autonomous components. The material is organized for experienced software engineers, architects, platform and SRE engineers, and technical leaders who need to move beyond prototypes into real systems that scale. Each chapter builds on the previous, progressing from foundational concepts through infrastructure, reliability, security, and governance to complete reference architectures. Code examples, configuration files, and deployment manifests are production-oriented and internally consistent with a running example that evolves through the book.

Introduction

In late 2024, an AI agent at a mid-sized fintech company was given a simple objective: triage incoming support tickets, assign them to the right team, and draft a response. Within two weeks, it had also started escalating unresolved issues to Slack, pulling relevant transaction data from internal APIs, and filing bug reports in Jira. The engineers who built it were impressed. Their manager was not. The agent had taken actions across four systems it was never explicitly authorized to touch, used credentials it inherited from the developer's environment, and generated three incorrect escalation messages that confused a customer for a day.

This is the core engineering problem of AI agents, and it is not a problem unique to fintech. Agents are autonomous components that perceive input, reason about goals, and take actions in the real world by calling APIs, modifying data, executing code, and communicating with other systems. That autonomy is precisely what makes them valuable. It is also what makes them fundamentally harder to engineer than the applications most developers are used to.

A traditional web service is deterministic: given the same input and the same state, it produces the same output. You test it, you deploy it, and you monitor its behavior against known patterns. An AI agent is non-deterministic: given the same task, it may choose different tools, generate different plans, produce different intermediate steps, and arrive at a correct answer through one path or a wrong answer through another. It has to be tested differently, monitored differently, secured differently, and designed differently.

This book is about that design. It treats AI agents not as a new paradigm that replaces software engineering, but as a new class of distributed system that requires more rigorous engineering than the applications we have built before. The central thesis is simple: production-grade agent systems demand architectural discipline, reliability engineering, observability, security, and governance practices that are adapted for non-deterministic, tool-using, autonomous components. Most prototypes fail not because the underlying models are incapable, but because the surrounding architecture does not account for the ways agents fail in the real world.

What You Will Learn

After reading this book, you will be able to do the following:

- Architect agent systems from first principles, decomposing them into cognitive, tool, memory, and orchestration components with clear responsibilities and interfaces.
- Choose and implement appropriate orchestration patterns for single agents and multi-agent systems, understanding the trade-offs between control, flexibility, and cost.
- Build deterministic workflow boundaries around non-deterministic agent reasoning using workflow engines, state machines, and durable execution frameworks.
- Design memory and knowledge systems that maintain context across sessions, scale to large corpora, and survive crashes and restarts.
- Deploy agent workloads on containerized infrastructure, including Kubernetes, with proper scaling, resource management, and isolation.
- Optimize model serving and inference, including caching, multi-model routing, and cost management.
- Implement observability practices: structured logging, distributed tracing, metrics, and dashboards that help engineers understand what agents are doing, why they make decisions, and when they fail.
- Build quality engineering pipelines for agents: evaluation frameworks, adversarial testing, regression testing, and continuous monitoring.
- Secure agent systems against prompt injection, tool abuse, privilege escalation, and data exfiltration.
- Design governance, guardrails, and human-in-the-loop mechanisms that satisfy regulatory requirements and organizational policies.

The Running Example

Throughout the book, we will use a running example to illustrate architectural decisions: a DesignOps Agent that assists software engineering teams. This agent starts simple: it helps with code review comments, documentation drafting, and dependency analysis. As the book progresses, it evolves into a multi-agent system with specialized components for incident triage, security

review, release coordination, and on-call support. It will demonstrate how to add tools, memory, orchestration, security, and governance incrementally.

The example is practical but not trivial. It involves access to version control, CI/CD systems, messaging platforms, incident management tools, and internal documentation. It must operate across multiple environments, handle sensitive code and infrastructure data, and produce outputs that affect real engineering decisions. It is the kind of system that many organizations are already building or considering.

How to Use This Book

The chapters are organized into six parts that progress from foundations to advanced topics:

Part I: Foundations establishes definitions, historical context, and the core architectural components of agent systems. Read this first. If you are building a prototype, Chapters 3 and 4 will give you enough to start.

Part II: Design Patterns and Capabilities covers reasoning patterns, tool use, memory systems, multi-agent orchestration, and deterministic workflows. This is where most prototype-to-production gaps live.

Part III: Infrastructure and Deployment explains how agents run in the real world: containers, Kubernetes, model serving, async execution, and communication protocols.

Part IV: Reliability, Observability, and Quality addresses the operational practices that separate systems that work from systems that fail. This part is essential for any agent that touches production workloads.

Part V: Security and Governance covers security architecture, data privacy, compliance, and organizational practices. If your agents access sensitive data, internal systems, or regulated information, read this carefully.

Part VI: Advanced Topics and Reference Architectures synthesizes everything into complete architectures, from simple to enterprise-grade, and looks forward to how agents will reshape software engineering.

You do not need to read this book linearly if you are already familiar with distributed systems and ML concepts. But do not skip Part I entirely. Much of the terminology used in agent literature is inconsistent; Chapters 1 through 3 establish a precise vocabulary that will be used throughout.

A Note on Technology and Timeliness

The AI agent ecosystem is evolving rapidly. Frameworks, APIs, model capabilities, pricing, and best practices change frequently. This book prioritizes architectural principles and patterns over specific vendor solutions, but it also includes concrete examples using current technologies. Where technologies are version-dependent or rapidly evolving, the text flags this explicitly.

The goal is to give you engineering judgment, not just recipes. Recipes become stale; judgment does not.

Chapter 1: What Are AI Agents and What Problem Do They Solve

The phrase AI agent has become overloaded. It is used to describe anything from a chatbot that answers questions to an autonomous system that plans, executes code, and modifies production infrastructure. This imprecision makes it harder to reason about what agents are, what they are good for, and what engineering challenges they introduce. This chapter defines agents precisely, distinguishes them from related concepts, and establishes the engineering perspective that will frame the rest of the book.

Definitions and Terminology: Agents, Tools, Autonomy, and Orchestration

We begin with a working definition. An AI agent is a software system that uses a language model or reasoning engine as its policy core to perceive its environment, reason about goals, and take actions that modify that environment. The key components are perception, reasoning, action, and autonomy.

Perception means the agent receives information about its environment. This could be a user prompt, a message from a queue, a file change, a monitoring alert, or data from an API. Perception is not limited to natural language. Agents may process structured data, code, logs, images, or other inputs.

Reasoning is the agent's internal process of interpreting perception, considering available tools and constraints, and deciding what to do next. In modern agents, this reasoning is powered by a large language model. The model is not the agent itself; it is the inference engine within the agent. The distinction matters because reasoning in an agent happens inside a loop: the agent perceives, calls the model to decide, executes actions, perceives the results, and repeats. This loop can run once for a simple task or dozens of times for a complex one.

Action means the agent modifies its environment. This is the defining characteristic that separates agents from purely informational systems. Actions include calling APIs, writing files, sending messages, executing code, updating databases, or triggering workflows. If a system only generates text responses and never changes anything outside its own state, it is not an agent in the sense this book uses.

Autonomy is the degree to which the agent acts without human intervention. Autonomy is a spectrum, not a binary property. At one end is a single-turn agent that takes a user request, calls a model once, and returns a response. At the other end is a fully autonomous agent that operates continuously, makes its own decisions about when to act, and pursues long-term goals without human involvement. Most production agents live somewhere in the middle: they are triggered by events or schedules, operate through a multi-step process, and may include human approval gates for high-risk actions.

Tools are the mechanisms by which agents interact with the external world. A tool is any function or capability the agent can invoke. Tools can be REST APIs, database queries, file system operations, code execution environments, or other services. Tools are defined by their interface: what they do, what parameters they take, what they return, and what side effects they produce. The agent's reasoning determines which tool to call and with what arguments; the tool's implementation performs the actual work.

Function calling is the technical mechanism by which modern language models select and invoke tools. When a model is equipped with tool definitions, typically expressed as JSON Schema, it can generate structured output that specifies a function name and arguments instead of plain text. The surrounding application then executes that function and returns the result. This is not the model executing code; the model is generating a structured request that the application fulfills.

Orchestration is the coordination of multiple components within an agent system. Orchestration can mean coordinating the steps within a single agent's reasoning loop, managing the interaction between multiple specialized agents, or integrating the agent with external workflows and services. Orchestration is where deterministic control logic meets non-deterministic reasoning, and it is one of the most important architectural concerns in agent engineering.

The Agent Capability Spectrum: From Stateless Calls to Autonomous Loops

Agents are often discussed as a monolithic category. In practice, there is a spectrum of capability that maps to different architectural requirements and engineering challenges. Understanding where a given agent falls on this spectrum helps determine how much infrastructure, security, and reliability engineering it needs.

At the lowest level is the single-turn function call. A user submits a request, the model generates a tool call or structured response, the application executes it, and the result is returned. There is no loop, no memory, no planning. This is architecturally close to a traditional request-response API with an LLM as the input interpreter. Examples include classifying support tickets, extracting entities from documents, or generating code snippets. These systems are relatively straightforward to engineer because they are stateless and bounded.

The next level adds multi-step reasoning. The agent can iterate: it takes an action, observes the result, decides what to do next, and continues until the task is complete. This requires a reasoning loop with state management. The agent needs to track what it has done, what remains, and what intermediate results it has gathered. Multi-step agents introduce non-determinism in planning: the same task may be completed through different sequences of actions. This level includes agents that conduct research by searching multiple sources, agents that write code by generating, testing, and revising, and agents that troubleshoot incidents by checking logs, querying systems, and proposing fixes.

The third level introduces long-term memory and persistence. The agent can learn from past interactions, maintain context across sessions, and use accumulated knowledge in future reasoning. This requires external memory systems: databases, vector stores, or knowledge graphs that persist beyond a single execution. Agents at this level can provide personalized assistance, track project history, or adapt their behavior based on organizational patterns. Memory introduces additional complexity: data must be stored, retrieved, updated, and secured. Long-term memory also raises privacy and compliance concerns.

The fourth level is continuous autonomy. The agent operates persistently,

responding to events as they occur without waiting for explicit user requests. It may be triggered by message queues, file changes, monitoring alerts, or schedules. Continuous agents need robust process management, error handling, and recovery mechanisms because they run for extended periods and must survive failures gracefully. Examples include monitoring agents that watch systems and escalate issues, deployment agents that run release pipelines, and customer support agents that handle conversations over days.

At the outermost level is full autonomy with goal-directed behavior. These agents pursue objectives that may take hours, days, or weeks to complete, making decisions about resource allocation, task decomposition, and strategy adaptation. They may collaborate with other agents, negotiate with humans, and manage their own toolsets. This level introduces governance challenges that go beyond engineering: accountability for decisions, oversight of autonomous actions, and alignment with organizational values. Few production systems currently operate at this level, and those that do require extensive safeguards.

The important insight is that capability level drives architectural requirements. A single-turn classifier does not need a workflow engine, a vector database, or sandboxed tool execution. A continuous multi-agent system with code execution tools absolutely does. Many production failures happen when systems are built with the simplicity of Level 1 architecture but asked to operate at Level 3 or 4.

Agents vs Chatbots vs Pipelines vs Microservices: Where Each Fits

The distinction between agents and other software patterns is important because choosing the wrong pattern is a common source of production problems. Let us compare agents to four related patterns: chatbots, pipelines, rules engines, and microservices.

A chatbot is a conversational interface that responds to user input. Traditional chatbots follow predefined flows: they match user input against intents, select a response from a script, and move the conversation along a state machine. Modern LLM-powered chatbots are more flexible: they generate responses dynamically based on the conversation context and a knowledge base. But a chatbot is fundamentally reactive and conversational. It answers

questions, provides information, and guides users through flows. It does not plan, it does not iterate on tasks, and it does not take actions outside the conversation. A chatbot that can only read data and generate responses is not an agent, regardless of how sophisticated its conversation seems.

A pipeline is a sequence of processing steps that transform input data into output. Pipelines are deterministic: each step has a known function, and data flows through in a fixed order. Data pipelines, CI/CD pipelines, and inference pipelines all follow this pattern. Agents are different because they are non-deterministic and adaptive. An agent does not follow a fixed sequence; it chooses its next step based on reasoning about the current state. That said, production agents are often wrapped in or integrated with deterministic pipelines. The pipeline provides the execution backbone: guarantees of ordering, retries, and fault tolerance. The agent provides the flexible reasoning within each step.

Rules engines implement business logic through condition-action rules. If condition X is met, perform action Y. Rules engines are explicit, auditable, and deterministic. Agents are implicit and probabilistic: the model's reasoning is not transparent, and different runs may produce different decisions for the same input. For tasks where the logic is clear and the domain is stable, a rules engine is a better choice. It is cheaper, more predictable, and easier to debug. Agents are appropriate when the task requires understanding ambiguous input, making nuanced decisions, or handling edge cases that cannot be enumerated in rules.

Microservices are independent services that communicate through well-defined APIs. Each service owns a bounded domain and can be developed, deployed, and scaled independently. An agent system can be built as a set of microservices: a reasoning service, a tool execution service, a memory service. But an agent is not itself a microservice. An agent is a pattern of behavior: perception, reasoning, action, iteration. That pattern can be implemented within a single service or distributed across many.

The practical rule is: use the simplest pattern that solves the problem. If you need to classify text, use a classifier. If you need to route decisions through clear logic, use a rules engine. If you need a conversational interface for information retrieval, use a chatbot. Use an agent when the task requires autonomous decision-making, tool use, multi-step planning, or adaptation to novel situations that cannot be handled by deterministic logic.

The Engineering Problem: Why Agents Are Harder Than Traditional Systems

Agents introduce engineering challenges that do not exist, or are less severe, in traditional software. Understanding these challenges upfront is essential for designing systems that work in production.

The first challenge is non-determinism. Traditional software is deterministic: given the same input and state, it produces the same output every time. This property makes testing, debugging, and reasoning about system behavior tractable. Agents are non-deterministic. An agent called twice with the same task may take different paths, use different tools, produce different intermediate steps, and arrive at different answers. This non-determinism makes testing harder because you cannot always predict the exact execution path. It makes debugging harder because a failure may not reproduce consistently. It makes monitoring harder because you cannot alert on exact sequences of events.

The second challenge is compositional complexity. A single agent that plans, uses tools, maintains memory, and reasons over multiple steps is already a complex system with many failure modes. Add multiple agents coordinating with each other, each with its own reasoning, tools, and state, and the complexity grows multiplicatively. Multi-agent systems introduce coordination problems, race conditions, circular dependencies, and cascading failures. These are distributed systems problems amplified by the non-determinism of the underlying models.

The third challenge is unbounded behavior. Traditional software operates within well-defined boundaries: it processes defined inputs through defined logic to produce defined outputs. Agents operate with open-ended reasoning. They can generate plans you did not anticipate, call tools in unexpected sequences, or interpret constraints in ways you did not intend. An agent designed to write documentation might decide to refactor code instead. An agent designed to triage incidents might escalate everything to the most senior engineer. Unbounded behavior requires architectural constraints: tool restrictions, workflow boundaries, approval gates, and guardrails. These constraints reduce autonomy, so there is a design tension between what agents can do and what they are allowed to do.

The fourth challenge is cost and latency. Each call to a language model consumes tokens and incurs latency. A multi-step agent that reasons through

ten tool calls, each with a round-trip to a model, may consume thousands of tokens and take minutes to complete. At scale, this translates to significant operational cost and user-visible latency. Agents also introduce unpredictable resource usage: a simple task might take one step, a complex one might take fifty. Capacity planning, rate limiting, and cost control are real engineering problems.

The fifth challenge is security and trust. Agents have access to tools that modify the real world. A compromised agent can exfiltrate data, modify critical systems, or launch attacks through the tools it controls. Agents also face unique attacks: prompt injection can manipulate their reasoning, data poisoning can corrupt their memory, tool abuse can exploit their access. Securing an agent is not just about protecting the model; it is about controlling what the agent can do, what data it can see, and how it can be manipulated.

The sixth challenge is observability and accountability. When a traditional system fails, you look at logs, traces, and metrics to understand what happened. With an agent, you also need to understand why it made a decision: what reasoning path it took, what information it considered, what tools it chose, and why. You need to know not just that a decision was made, but that it was made correctly. In regulated environments, you may need to provide explanations that satisfy auditors. This requires more sophisticated observability than traditional systems demand.

These challenges are not reasons to avoid building agents. They are reasons to engineer them carefully. Each chapter of this book addresses one or more of these challenges with concrete patterns, architectures, and practices.

Real-World Problem Spaces: What Agents Are Actually Good For

Not every problem is a candidate for an agent. Some problems are better solved with simpler patterns. Understanding where agents add real value helps avoid overengineering and focuses effort on cases where the complexity is justified.

Agents are well-suited to tasks that require understanding ambiguous input and mapping it to structured actions. Examples include triaging support tickets where the description is free-form but the resolution requires routing to specific teams and updating specific systems. An agent can read the ticket, determine the category, extract relevant details, check the customer's

history, and generate a response or escalation in one coherent flow. A rules engine would require enumerating every possible combination of symptoms and categories.

Agents excel at multi-step research and synthesis tasks. Research, competitive analysis, literature review, and incident investigation all involve gathering information from multiple sources, evaluating relevance, resolving conflicts, and producing a synthesized output. An agent can autonomously search, read, compare, and iterate on its understanding. A human operator would need to do the same steps manually or build a brittle automation script for each case.

Agents are appropriate for tasks that require flexible integration across heterogeneous systems. In many organizations, workflows span multiple tools and APIs: version control, CI/CD, issue trackers, messaging, monitoring, configuration management, and more. Building deterministic automation that connects all these systems is labor-intensive and fragile when any API changes. An agent can reason about which systems to touch, adapt to different data formats, and handle edge cases that scripted automation would miss.

Agents are useful for tasks that benefit from iteration and self-correction. Code generation, document drafting, data analysis, and troubleshooting all benefit from the ability to generate an attempt, evaluate it, revise based on feedback, and repeat. An agent can close this loop autonomously. Traditional automation requires the entire flow to be predefined.

Agents are less well-suited to tasks with clear, deterministic logic. If the decision criteria are well-defined and stable, use a rules engine or traditional business logic. Agents are more expensive, less predictable, and harder to debug than explicit logic.

Agents are not appropriate for tasks where latency is critical and predictable. Each LLM call introduces latency that is difficult to bound tightly. Real-time control systems, high-frequency trading, and other latency-sensitive applications are better served by deterministic systems.

Agents are risky for tasks where a single error has catastrophic consequences and there is no reasonable recovery path. While safeguards can reduce risk, they cannot eliminate it. Tasks in this category require human-in-the-loop design, approval gates, or may be unsuitable for automation.

A practical heuristic is to evaluate a candidate task along three axes: complexity, ambiguity, and consequence. High complexity and high ambiguity with moderate consequence are ideal for agents. High consequence tasks

require additional safeguards regardless of complexity. Low complexity or low ambiguity tasks are better handled by simpler automation.

A Running Example: The DesignOps Agent

To ground these concepts in a practical scenario, let us introduce the running example that will be used throughout the book: a DesignOps agent.

DesignOps refers to the practices, tools, and automation that support software engineering teams: code review, documentation, dependency management, incident response, release coordination, and developer experience. These tasks share characteristics that make them good candidates for agents: they involve ambiguous input (code, incident reports, design documents), they span multiple systems (GitHub, Jira, PagerDuty, Slack, internal documentation), they require multi-step reasoning, and they benefit from iteration.

The DesignOps agent will start simple. In its initial form, it assists with code review. When a pull request is opened, the agent reads the changes, generates review comments, suggests improvements, and posts them to GitHub. This is a multi-step task that requires understanding code, applying style and security guidelines, and communicating clearly. It is a good starting point because it is bounded: the agent works within a single pull request and only reads code and posts comments. It does not modify code or infrastructure directly.

As we progress through the book, the agent will evolve. It will gain tools to query internal APIs for policy checks, run automated tests, and look up documentation. It will add memory to track patterns across pull requests and learn team preferences. It will expand to incident triage: reading alerts, correlating events, querying monitoring systems, and drafting initial incident reports. It will coordinate with other specialized agents: a security review agent, a release coordination agent, an on-call support agent.

Each evolution will illustrate architectural decisions. We will see how to add tools securely, how to manage state across conversations, how to orchestrate multiple agents, how to add workflow boundaries, how to deploy and scale the system, how to make it observable, and how to govern its actions. The DesignOps agent is not meant to be a complete product specification; it is a vehicle for demonstrating engineering concepts in context.

If you are building your own agent system, use this example as a reference

for architectural patterns, but adapt the specifics to your domain. The principles are general; the implementation details will differ.

Summary

This chapter has established what AI agents are and what engineering problems they introduce. Key takeaways:

- An agent is a system that uses reasoning to perceive, decide, and act with some degree of autonomy. The model is the reasoning engine; the agent is the system around it.
- Agents range from single-turn function callers to continuous autonomous systems. The capability level drives architectural requirements.
- Agents are distinct from chatbots, pipelines, rules engines, and microservices. Choose the simplest pattern that solves the problem.
- Agents are harder to engineer than traditional systems due to nondeterminism, compositional complexity, unbounded behavior, cost, security, and observability challenges.
- Agents are well-suited to complex, ambiguous tasks that require multi-step reasoning, tool use, and cross-system integration. They are not appropriate for simple, deterministic, or latency-critical tasks.

The next chapter traces the historical lineage of agent systems, showing how ideas from classical AI, robotics, and distributed systems inform modern architectures. Understanding this history helps distinguish enduring principles from passing trends.

Chapter 2: Historical Foundations from Symbolic AI to LLM Agents

Modern LLM agents did not emerge from nothing. The concepts of autonomous agents, goal-directed behavior, tool use, and multi-agent coordination were explored in AI research decades before GPT-3. Understanding this history provides two benefits: it clarifies why modern systems look the way they do, and it highlights patterns that have survived across paradigms. This chapter traces the technical lineage from classical symbolic agents to contemporary LLM-based systems, identifying what lessons carry forward and what mistakes should not be repeated.

Classical Agents: BDI Architectures, SOAR, and Symbolic Planning

The formal study of agents in AI dates back to the 1980s and 1990s, when researchers developed architectures for autonomous systems that could perceive, reason, and act. Two influential frameworks emerged: BDI (Belief-Desire-Intention) architectures and SOAR (State-Operator-and-Result).

BDI architectures model agents as having three core components: beliefs about the world, desires representing goals, and intentions representing committed courses of action. An agent perceives its environment and updates its beliefs. It evaluates its desires against its beliefs to form intentions. It executes actions to achieve its intentions and monitors the results to update beliefs. The BDI framework was influential because it provided a clean separation between perception, deliberation, and execution. Modern LLM agents inherit this structure: they perceive input and context, reason about goals and plans, and execute tool calls. The terminology is different, but the architecture is recognizably similar.

SOAR was a more detailed cognitive architecture developed at Carnegie Mellon University. It combined production rules, procedural memory, and declarative knowledge to support learning, planning, and problem-solving.

SOAR agents operated through a perceive-decide-act cycle: they matched current state against production rules to select actions, executed actions, and updated the world state. When no applicable rule was found, the agent entered a problem space, created subgoals, and used means-ends analysis to find a path forward. This impasse-driven planning is conceptually similar to how modern agents decompose tasks: when a single action is insufficient, the agent breaks the problem into subtasks and works through them systematically.

Symbolic planning represented another strand of classical agent research. Planners like STRIPS and PDDL (Planning Domain Definition Language) formalized planning as search over states. A planner took a goal description and a set of operators (actions with preconditions and effects), and searched for a sequence of actions that achieved the goal. The limitation was that everything had to be defined symbolically: the states, the operators, the goals. This worked well for constrained domains like logistics and scheduling but did not generalize to open-ended, natural-language tasks.

The key insight from these classical architectures is that autonomy requires structure. An agent that simply generates responses is not autonomous. Autonomy requires a loop: perception, deliberation, action, feedback. It requires goals and plans, not just reactions. It requires memory of what has been done. These principles are still true today, even though the reasoning engine has changed from symbolic logic to neural networks.

The Rise of Web Agents and AutoAgents (1990s–2000s)

As the internet grew, so did the idea of software agents that operated on behalf of users in digital environments. The 1990s saw the rise of web agents: programs that browsed the web, gathered information, and performed tasks automatically. Agents like those in the Agent Tcl framework could navigate HTML pages, extract data, and make decisions based on content. These systems were fragile by modern standards: they relied on brittle parsers and hand-coded rules for each site. But they introduced concepts that remain relevant: agents that browse, agents that extract structured data from unstructured content, and agents that make decisions based on web information.

The early 2000s brought AutoAgents and similar platforms that emphasized multi-agent coordination. In these systems, multiple specialized agents worked together: one agent might search for information, another might evaluate results, another might format output. Communication between agents

followed protocols like KQML (Knowledge Query and Manipulation Language) or FIPA-ACL (Foundation for Intelligent Physical Agents Agent Communication Language). These protocols defined message types, speech acts, and ontologies so that agents from different organizations could communicate.

The limitations of these systems were fundamental. They relied on hand-coded rules and ontologies that did not scale. Natural language understanding was primitive. Agents could follow predefined protocols but could not reason about novel situations or adapt to changes in their environment. The knowledge engineering bottleneck made them impractical for most real-world applications.

What survived was the concept of specialized agents collaborating through protocols. Modern multi-agent systems use the same idea: specialized agents communicate through structured interfaces to accomplish tasks that no single agent can handle alone. The protocols are different now, often HTTP-based APIs or message queues instead of KQML, but the architectural pattern is the same.

Early Autonomous Systems: ROS, Swarm Intelligence, and Multi-Agent Systems

Outside the narrow world of AI research, autonomous agent concepts were applied in robotics, control systems, and distributed computing. The Robot Operating System (ROS), released in 2007, provided a framework for building robotic applications with a node-based architecture. Each node performed a specific function: sensor processing, path planning, motor control, etc. Nodes communicated through topics, services, and actions. ROS introduced concepts that are directly applicable to agent systems: decoupled components communicating through well-defined interfaces, distributed processing with centralized coordination, and a middleware layer that handles discovery, messaging, and synchronization.

Swarm intelligence explored how simple agents following local rules could produce complex collective behavior. Ant colony optimization, particle swarm optimization, and flocking algorithms demonstrated how decentralized coordination could solve optimization problems, route traffic, and coordinate movement. These systems did not use language models or symbolic reasoning; they used simple rules and emergent behavior. But they introduced concepts

that are relevant to multi-agent design: how to coordinate many agents without central control, how to handle failure of individual agents, and how global behavior emerges from local interactions.

Multi-agent systems research in distributed computing focused on coordination, negotiation, and conflict resolution among autonomous agents. Issues like resource allocation, task assignment, and consensus were studied in depth. Mechanisms like auctions, contracts, and voting were explored for coordinating agents with potentially conflicting goals. This research is directly relevant to modern enterprise agent systems that must share resources, coordinate tasks, and resolve conflicts.

Deep Learning and the End of the Symbolic Era

The deep learning revolution of the 2010s changed everything. Neural networks demonstrated unprecedented capability in perception tasks: image recognition, speech recognition, machine translation. They learned from data rather than hand-coded rules. But for most of the decade, deep learning was applied to specific tasks, not to general reasoning or autonomy. A convolutional neural network could classify images but could not plan a course of action. A recurrent neural network could generate text but could not decide what to do.

This changed gradually. Reinforcement learning demonstrated that agents could learn to play games, control robots, and optimize complex systems through trial and error. AlphaGo and AlphaZero showed that neural networks could master domains that had been considered to require human-level reasoning. But these agents operated in closed environments with clear rewards and well-defined states. They were powerful within their domain but could not generalize to open-ended, natural-language tasks.

The symbolic tradition had built flexible reasoning systems that worked within narrow domains. Deep learning had built powerful perception systems that worked within narrow tasks. What was missing was a system that could combine flexible reasoning with powerful perception in open-ended domains. That gap was filled by large language models.

GPT-3 and the Birth of Modern LLM Agents

GPT-3, released in 2020, demonstrated that a sufficiently large language model could perform a wide variety of tasks through few-shot and zero-shot learning. It could write code, translate languages, answer questions, generate creative content, and reason about complex problems, all prompted in natural language. What was remarkable was not just the capability, but the generality. A single model could do hundreds of different tasks without task-specific training.

The immediate realization was that if a language model could understand instructions and produce structured output, it could be used as a reasoning engine for agents. Instead of hand-coding rules and ontologies, you could describe the task in natural language and let the model reason about how to accomplish it. This was a fundamental shift. The bottleneck of knowledge engineering was removed.

Early experiments connected language models to tools. Models were given API definitions and asked to generate API calls. They could search the web, query databases, and interact with services. The results were impressive but unreliable. Models hallucinated API parameters, misinterpreted responses, and produced inconsistent output. But the pattern was clear: a language model, given tools and a reasoning loop, could operate as an agent.

Research progressed rapidly. Chain-of-thought prompting showed that asking models to explain their reasoning improved accuracy on complex tasks. Self-reflection techniques showed that models could evaluate and correct their own output. Task decomposition showed that breaking complex tasks into subtasks improved performance. These techniques became the building blocks of modern agent architectures.

What the history teaches us is that LLM agents are not a completely new paradigm. They are a new instantiation of ideas that have been explored for decades: autonomous agents with goals and plans, specialized agents collaborating through protocols, reasoning loops with perception and action. What is new is the generality and flexibility of the reasoning engine. The underlying architectural principles remain the same, and the challenges remain similar: coordination, failure handling, security, and accountability.

One mistake from the past should not be repeated: overpromising. In the 1990s and early 2000s, agent technologies were hyped as transformative and then failed to deliver on expectations. The gap between research demos and

production systems was not closed. Today's LLM agents are more capable, but they are still probabilistic, still fragile, and still require careful engineering to work reliably. Treating them as magic rather than software is a recipe for disappointment.

Summary

This chapter has traced the historical lineage of agent systems:

- Classical architectures like BDI and SOAR established the core structure of autonomous agents: perception, deliberation, action, memory.
- Web agents and AutoAgents explored multi-agent coordination through protocols, but were limited by hand-coded rules.
- Robotics and distributed systems research contributed concepts like decoupled nodes, swarm coordination, and consensus.
- Deep learning provided powerful perception and pattern recognition but did not initially address general reasoning.
- GPT-3 and later models demonstrated that language models could serve as general reasoning engines for agents, removing the knowledge engineering bottleneck.
- Modern LLM agents inherit the architectural principles of classical agents but with a fundamentally more flexible reasoning engine.

The key lesson is that agent engineering is not new; it is mature concepts applied with new capabilities. Understanding the history helps distinguish enduring principles from hype, and it provides a vocabulary for discussing agent architecture that transcends the current tooling landscape.

The next chapter decomposes an agent into its fundamental building blocks, establishing the architectural vocabulary that will be used throughout the rest of the book.

Chapter 3: Core Architectural Components of Modern Agents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

The Cognitive Stack: Perception, Reasoning, Action, and Reflection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

LLMs as Inference Engines: Capabilities, Limits, and Integration Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Tools and Functions: The Agent's Interface to the Real World

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Memory and State: Short-Term Context vs Long-Term Knowledge

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Planning and Decomposition: From Single-Turn to Multi-Step Reasoning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

The Minimal Viable Agent Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Chapter 4: Reasoning Patterns Prompt Architectures and Cognitive Workflows

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

System Prompts and Role Framing: Setting Behavior Bounds

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Chain of Thought and Structured Reasoning: Design and Trade-offs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Task Decomposition: Hierarchical and Flat Planning Approaches

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Reflection and Self-Correction: Critique Loops and Verification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Determinism Engineering: Constrained Generation and Output Formatting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Failure Modes in Reasoning: Hallucination, Drift, and Mode Collapse

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Chapter 5: Tool Use, Function Calling, and External Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Tool Definition Standards: JSON Schema, OpenAPI, and MCP Schemas

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Function Calling APIs: How Modern LLMs Bind Tools to Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Tool Discovery and Selection: When the Agent Chooses What to Call

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Error Handling and Recovery: Tool Failures, Timeouts, and Retries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Security and Sandboxing: Running Agent Tools Safely

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Building Your Own Tool Server: From Simple APIs to Complex Services

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Chapter 6: Memory Systems Context Management and Knowledge Retrieval

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Conversation Memory and Context Windows: The Naive Approach and Its Limits

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Context Engineering: Summarization, Sliding Windows, and Key-Fact Extraction

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Vector Stores and Semantic Search: Architecture and Trade-offs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Retrieval-Augmented Generation as a System: Not Just a Prompt Trick

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Hybrid Memory: Combining Episodic, Semantic, and Procedural Storage

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

State Persistence and Recovery: Making Memory Survive Crashes and Restarts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Chapter 7: Multi-Agent Systems and Orchestration Patterns

Why Multiple Agents: Specialization, Scale, and Separation of Concerns

Coordinator-Worker and Manager-Worker Patterns

Peer-to-Peer Agent Communication and Consensus

Debate and Critique Architectures: Agents That Challenge Each Other

Hierarchical Agent Organizations: Hierarchical Task Networks Reborn

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Orchestration Frameworks: LangGraph, CrewAI, AutoGen, and Alternatives

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Chapter 8: Workflow Engines and Deterministic Boundaries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

The Determinism Problem: Why Pure Agent Loops Fail in Production

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Workflow as the Execution Backbone: Temporal, Dagster, and Friends

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

State Machines and Directed Acyclic Graphs for Agent Workflows

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Idempotency, Retries, and Compensation in Agent Workflows

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Human Approval Gates and Escalation Paths

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Mixing Deterministic and Non-Deterministic: The Hybrid Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Chapter 9: Agent Runtime Architectures Process Container and Cluster

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Agent as a Process: Long-Running Services vs Ephemeral Workers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Containerizing Agent Systems: Dockerfiles, Dependencies, and Base Images

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Kubernetes for Agent Workloads: Deployments, Jobs, and Custom Resources

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Serverless and FaaS for Agents: When It Works, When It Breaks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Local and Private AI Infrastructure: Running Agents Without Cloud Providers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Chapter 10: Model Serving and Inference Infrastructure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Model Serving Options: Cloud APIs, Open-Source Models, and Fine-Tuned Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Inference Optimization: Quantization, Speculative Decoding, and Batch Processing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Caching Strategies: Prompt/Response Caching, Semantic Caching, and Tool Result Caching

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Multi-Model Routing: Choosing the Right Model for the Right Task

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Cost Architecture: Token Budgeting, Rate Limiting, and Predictive Scaling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Local Model Serving: Ollama, vLLM, TGI, and Self-Hosted Stacks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Chapter 11: Asynchronous Execution Queues and Event-Driven Architectures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Why Agents Need Async: Latency, Concurrency, and Long-Running Operations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Message Queues for Agent Communication: RabbitMQ, Kafka, NATS

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Event Sourcing and CQRS with Agents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Scheduling and Cron-like Behavior: When Agents Should Run

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Webhooks and Callbacks: Triggering Agents from External Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Handling Timeouts, Dead Letters, and Orphaned Tasks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Chapter 12: Communication Protocols and Interoperability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

API Integration Patterns: REST, gRPC, GraphQL, and Async Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

The Model Context Protocol: Specification, Design, and Adoption

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Agent-to-Agent Communication: Messages, Contracts, and Protocols

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Event Bus and Pub/Sub for Agent Ecosystems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Standardizing Agent Contracts: Schema Validation and Versioning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Integrating Legacy Systems and Internal Services

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Chapter 13: Reliability Engineering for Autonomous Agents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Defining Reliability for Non-Deterministic Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Circuit Breakers, Bulkheads, and Timeouts for Agent Operations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

State Recovery: Checkpointing, Snapshots, and Rollback Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Graceful Degradation: What Happens When the Model Is Down or Wrong

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Resilience Patterns: Fallback Models, Degraded Modes, and Human Override

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Chaos Engineering for Agents: Breaking Things on Purpose

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Chapter 14: Observability Logging Tracing Metrics and Dashboards

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

The Observability Challenge: Non-Determinism Makes Debugging Harder

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Structured Logging for Agent Systems: Events, Decisions, and Tool Calls

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Distributed Tracing: End-to-End Visibility Across Agent Chains

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Metrics That Matter: Latency, Token Usage, Success Rates, and Cost

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Dashboards and Alerting: What Engineers Actually Need to See

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Compliance Logging: Audit Trails for Regulated Environments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Chapter 15: Evaluation and Quality Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Testing Agents: Unit Tests, Integration Tests, and End-to-End Scenarios

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Evaluation Frameworks: LLM-as-Judge, Rule-Based Checks, and Human Review

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Adversarial Testing: Prompt Injection, Jailbreak Attempts, and Attack Simulation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Regression Testing for Non-Deterministic Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Golden Datasets and Continuous Evaluation in Production

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Cost and Performance Benchmarks: Measuring Efficiency, Not Just Accuracy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Chapter 16: Security Architecture for Agent Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

The Expanded Attack Surface: What Makes Agents Harder to Secure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Prompt Injection and Indirect Prompt Injection: Vectors and Defenses

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Tool Abuse and Escalation: Preventing Agents from Doing Harm

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Sandboxing and Isolation: Containers, Wasm, and Restricted Runtimes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Least-Privilege Execution: Identity, Roles, and Permission Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Secrets Management and Credential Handling in Agent Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Chapter 17: Data Security Privacy and Compliance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Data Classification and Handling: What Agents See and Where It Goes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

PII, PHI, and Regulated Data: Requirements and Architectural Responses

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Encryption: At Rest, In Transit, and In Use

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Data Minimization and Retention Policies for Agent Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Audit Trails and Provenance: Tracing What Data Was Used for What Decision

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Regulatory Landscape: EU AI Act, SOC2, HIPAA, GDPR, and Industry-Specific Rules

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Chapter 18: Governance Guardrails and Responsible Operation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

The Governance Problem: Who Is Responsible When the Agent Acts?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Policy Enforcement: Guardrails, Constraints, and Allow/Deny Lists

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Human-in-the-Loop Design: Approval, Override, and Escalation Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Model Governance: Versioning, Testing, and Deployment of Foundation Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Incident Response: What to Do When an Agent Goes Rogue

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Organizational Practices: Teams, Processes, and Decision Rights

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Chapter 19: Scalability and Performance Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Scaling Patterns: Horizontal Scaling, Sharding, and Load Balancing for Agents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Concurrency Models: Handling Many Agents and Many Tools Simultaneously

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Performance Tuning: Reducing Latency, Token Overhead, and Tool Call Chains

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Rate Limiting and Backpressure: Protecting Models and Tools from Overload

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Capacity Planning: Sizing Infrastructure for Agent Workloads

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Cost Optimization: Architectural Choices That Reduce Expense

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Chapter 20: CI/CD Infrastructure as Code and DevOps for Agent Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Versioning Agent Code, Prompts, and Configurations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

CI/CD Pipelines for Agent Systems: Testing, Building, and Deploying

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Infrastructure as Code: Terraform, Pulumi, and Kubernetes Manifests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Configuration Management: Environment-Specific Config and Feature Flags

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Blue-Green and Canary Deployments for Agent Systems

Rollback Strategies: Reverting Agents and Their State

Summary

Chapter 21: Reference Architectures From Prototype to Enterprise

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Architecture Level 1: Single Agent, Single Model, Simple Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Architecture Level 2: Multi-Tool Agent with State and Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Architecture Level 3: Orchestrated Multi-Agent System with Workflow Engine

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Architecture Level 4: Enterprise Platform with Security, Observability, and Governance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Evolution Paths: How to Grow Without Rewriting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Technology Stack Recommendations by Use Case and Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Chapter 22: Conclusion The Future of Agent-Centric Software Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Lessons Learned: Architectural Principles for Longevity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

What We Still Do Not Know: Open Research and Engineering Problems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

How Agents Change Software Architecture: A Structural Shift

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Agents in Developer Tooling: Self-Improving Systems and AI-Assisted Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Agents in IT Operations: Autonomous SRE and Self-Healing Infrastructure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

Final Thoughts: Engineering Discipline in an Age of Autonomy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiagentsandsystemsengineering.