Faysal Aziz

Claude for Builders and Technical Teams

From AI Productivity to API-Driven Workflows, Coding Agents, MCP, and Enterprise Integration - 2 days

FA-0129 · 2 days

Agentic & Generative AI · Software Development · DevOps, Cloud & Infrastructure · Cybersecurity

---

Introduction

Claude becomes much more powerful when it moves beyond the chat window and into the systems where real work happens: code repositories, APIs, internal tools, databases, cloud platforms, and controlled team workflows. This two-day course is designed for participants who already understand Claude as a productivity assistant and are now ready to use it in more technical, structured, and operational ways. The foundation course emphasized writing, summarizing, planning, analysis, and responsible day-to-day use; this program extends that foundation into development workflows, API usage, tool calling, automation, Claude Code, MCP, and production-aware implementation.

The course is led by an instructor with over 30 years of industry experience and is built around real industry-demanded content rather than academic theory. The emphasis is practical: how developers, analysts, automation teams, and technically capable professionals can use Claude safely and effectively to build workflows, accelerate coding tasks, connect external systems, and design AI-assisted solutions that can survive real workplace constraints. Anthropic's current learning paths include Claude Platform, Claude Code, Building with the Claude API, MCP, subagents, skills, Bedrock, and Vertex AI, making these areas the natural next step for a more technical continuation.

Learning outcomes

- Understand how Claude fits into technical workflows beyond ordinary prompting
- Use the Claude API for structured application-level interactions
- Understand authentication, request structure, response handling, tokens, limits, and model selection

- Design better technical prompts for APIs, automation, and repeatable workflows
- Use tool calling concepts to connect Claude with external functions, systems, and APIs
- Understand client-side tools, server-side tools, and MCP-based integration patterns
- Use Claude Code concepts for AI-assisted software development workflows
- Apply context management, project memory, custom commands, hooks, skills, and subagents
- Understand MCP architecture, servers, clients, tools, resources, and prompts
- Identify when to use Claude directly, through cloud platforms, through coding agents, or through MCP
- Apply security, privacy, governance, and review practices for technical AI usage
- Plan practical adoption patterns for teams building with Claude in professional environments

Prerequisites

- Completion of a basic Claude productivity course or equivalent hands-on Claude usage
- General understanding of prompts, context, iteration, and responsible AI review
- Basic coding knowledge in Python, JavaScript, TypeScript, or a similar language
- Basic familiarity with APIs, JSON, HTTP requests, and response structures
- Comfort using a code editor such as VS Code
- Basic command-line or terminal usage
- Basic Git and version control awareness
- General understanding of cloud or web application concepts
- A Claude account, Claude API key, or access to an approved organizational Claude environment
- Awareness of basic data privacy, credential handling, and workplace security practices

Target Audience

- Software developers beginning to adopt Claude in development workflows
- Technical analysts and automation specialists
- Data professionals who need AI-assisted scripting and workflow automation
- DevOps, platform, and cloud engineers exploring Claude integration
- Solution architects and technical leads evaluating Claude for internal applications
- Product teams working with AI-enabled features
- IT professionals responsible for controlled AI adoption
- Power users who are ready to move from prompt-based productivity into APIs, tools, and automation
- Teams preparing to build internal AI assistants, coding workflows, or MCP-connected systems

Training outline

1. Technical Foundations for Building with Claude

Moving from productivity use to technical implementation

1. Claude as an assistant
2. Claude as an API
3. Claude as a coding agent
4. Claude as a connected workflow layer
5. Claude in enterprise and team environments

Claude technical ecosystem overview

1. Claude.ai
2. Claude Platform
3. Claude API
4. Claude Code
5. Claude Code SDK
6. Claude Managed Agents
7. Model Context Protocol
8. Skills
9. Subagents
10. Cloud platform access models
11. Direct API versus cloud provider access

Model selection and capability awareness

1. Claude model families
2. Capability differences across models
3. Latency and cost considerations
4. Reasoning and coding considerations
5. Context window considerations
6. Output length considerations
7. Feature availability by platform
8. Model lifecycle and version awareness

Technical AI collaboration mindset

1. Human-controlled implementation
2. AI-assisted development boundaries
3. Review-first engineering habits
4. Structured task decomposition
5. Systematic context preparation
6. Repeatable technical workflows
7. Responsible automation limits

2. Claude Platform and API Fundamentals

Claude Platform orientation

1. Console access
2. Workbench usage
3. API key management
4. Workspace organization
5. Usage monitoring
6. Spend control concepts
7. Team access considerations

API architecture fundamentals

1. Messages API
2. Message Batches API
3. Token Counting API
4. Models API
5. Files API
6. Skills API
7. Agents API
8. Sessions API
9. Environments API
10. Beta feature access
11. API versioning

Authentication and access control

1. API keys
2. Bearer token authentication
3. Workload identity concepts
4. Required request headers
5. SDK-managed headers
6. Workspace-based separation
7. Key rotation practices
8. Secret storage considerations
9. Environment variable usage

Request and response structure

1. Message roles
2. User messages
3. Assistant messages
4. System instructions
5. Content blocks
6. Text blocks
7. Tool use blocks
8. Tool result blocks
9. Stop reasons
10. Usage metrics
11. Request identifiers
12. Error response patterns

Messages API workflow

1. Stateless conversation handling
2. Full conversation history management
3. Multi-turn interaction design
4. Synthetic assistant messages
5. System prompt control
6. Output shaping
7. Structured response requirements
8. Prompt and response validation

9. Request size limits
10. Response parsing patterns

SDK-based development

1. Python SDK
2. TypeScript SDK
3. Client initialization
4. Request construction
5. Response handling
6. Retry behavior
7. Timeout handling
8. Streaming support
9. Error handling
10. Local environment setup

3. Prompt Engineering for Technical Systems

Prompt design for API workflows

1. Instruction hierarchy
2. System prompt design
3. Developer-facing prompt patterns
4. User input separation
5. Context packaging
6. Constraints and output contracts
7. Deterministic formatting requirements
8. Safety and refusal-aware prompts

Structured outputs

1. JSON response design
2. Schema-aligned prompting
3. Field-level output control
4. Response validation
5. Parsing reliability
6. Error-resilient output design

7. Downstream application compatibility

Technical task decomposition

1. Problem framing
2. Requirements extraction
3. Design planning
4. Implementation planning
5. Code generation prompting
6. Test generation prompting
7. Review prompting
8. Documentation prompting
9. Refactoring prompting

Context management for applications

1. Context window planning
2. Relevant context selection
3. Conversation history trimming
4. Compaction concepts
5. Context editing concepts
6. Long-document context handling
7. File-based context handling
8. Token counting
9. Prompt caching
10. Cost-aware context design

Reliability and quality controls

1. Output review checkpoints
2. Hallucination reduction practices
3. Source grounding patterns
4. Validation before execution
5. Confidence and uncertainty handling
6. Human approval loops
7. Logging and traceability
8. Reproducibility considerations

4. Tool Use and Function Calling with Claude

Tool use concepts

1. Connecting Claude to external tools and APIs
2. Client-side tools
3. Server-side tools
4. User-defined tools
5. Anthropic-defined tools
6. Tool descriptions
7. Tool schemas
8. Tool execution boundaries
9. Tool result handling

Tool calling workflow

1. Tool definition
2. Tool selection
3. Tool use response handling
4. Application-side execution
5. Tool result submission
6. Multi-step tool loops
7. Parallel tool use concepts
8. Stop reason handling
9. Error handling in tool loops

Schema design for tools

1. Input schema structure
2. Required fields
3. Optional fields
4. Type definitions
5. Parameter descriptions
6. Strict tool use
7. Schema validation
8. Tool naming conventions
9. Tool discoverability

Tool choice control

1. Automatic tool choice
2. Required tool use
3. Specific tool selection
4. No-tool responses
5. Prompt-based tool steering
6. Conservative tool invocation
7. Aggressive tool invocation
8. Tool fallback behavior

Server tools and built-in capabilities

1. Web search tool
2. Web fetch tool
3. Code execution tool
4. Tool search tool
5. Advisor tool concepts
6. Server-side execution boundaries
7. Usage-based cost considerations
8. Source citation considerations

Production considerations for tool use

1. Secure function execution
2. Input sanitization
3. Output validation
4. Rate limiting
5. Timeout handling
6. Idempotency
7. Observability
8. Audit logging
9. Failure recovery
10. Human approval gates

5. Building Claude-Powered Applications

Application architecture patterns

1. Direct API applications
2. Backend orchestration layer
3. Chat-based assistants
4. Workflow assistants
5. Document-processing assistants
6. Research assistants
7. Coding assistants
8. Tool-using agents
9. Human-in-the-loop systems

Backend integration patterns

1. REST API integration
2. Internal service integration
3. Database-aware workflows
4. File-processing workflows
5. Queue-based processing
6. Batch processing
7. Event-driven usage
8. Webhook-driven usage
9. Enterprise system integration

Conversation and session design

1. Stateless API handling
2. Session state management
3. User context storage
4. Project context storage
5. Memory boundary design
6. Role-based context access
7. Multi-user workflow considerations
8. Sensitive context separation

Data handling and file workflows

1. File upload concepts
2. Document parsing considerations
3. PDF support concepts
4. Image and vision workflows
5. Large file constraints
6. Reusable file references
7. File retention awareness
8. Data minimization practices

Cost and performance management

1. Token usage analysis
2. Prompt length control
3. Prompt caching strategy
4. Batch processing strategy
5. Latency trade-offs
6. Model selection by task
7. Streaming for user experience
8. Monitoring usage patterns
9. Cost governance

Testing and evaluation

1. Functional testing
2. Prompt regression testing
3. Tool-use testing
4. Structured output testing
5. Edge case testing
6. Safety testing
7. Human evaluation
8. Golden dataset concepts
9. Continuous improvement loops

6. Claude Code for AI-Assisted Development

Claude Code fundamentals

1. AI coding agent concepts
2. Difference between chat tools and coding agents
3. Agentic development loops
4. Tool integration in coding workflows
5. Codebase navigation
6. Code modification workflows
7. Development environment integration

Claude Code setup and usage

1. Terminal setup
2. Code editor integration
3. VS Code integration
4. JetBrains integration
5. Claude Desktop integration
6. Web usage considerations
7. Account and API key requirements
8. Permission models

Daily development workflows

1. Explore workflow
2. Plan workflow
3. Code workflow
4. Commit workflow
5. Feature implementation
6. Bug investigation
7. Refactoring
8. Documentation updates
9. Test creation
10. Code review support

Context management in Claude Code

1. Context window awareness
2. Project context selection
3. /compact
4. /clear
5. /context
6. File references
7. Directory references
8. Keeping sessions focused
9. Reducing irrelevant context
10. Context recovery practices

Project memory and instruction files

1. CLAUDE.md purpose
2. Repository-level instructions
3. Coding standards
4. Architecture notes
5. Testing instructions
6. Build commands
7. Review expectations
8. Maintenance practices

Planning and permission modes

1. Approval mode
2. Auto-accept mode
3. Plan Mode
4. Safe execution practices
5. Command approval boundaries
6. Risk-aware automation
7. Human decision checkpoints

Code review and Git workflows

1. Git repository awareness

2. Branch workflows
3. Commit preparation
4. Pull request preparation
5. Diff review
6. Automated review concepts
7. GitHub integration
8. Team review boundaries

7. Extending Claude Code

Custom commands

1. Command creation
2. Reusable development actions
3. Repository-specific commands
4. Team-level command patterns
5. Naming standards
6. Maintenance practices

Hooks

1. Hook concepts
2. Lifecycle events
3. Deterministic control
4. Formatting hooks
5. Command blocking hooks
6. Notification hooks
7. Validation hooks
8. Prompt-based hooks
9. Agent-based hooks
10. HTTP hooks
11. Async hooks
12. Hook configuration
13. Hook troubleshooting

Skills

1. Skill purpose
2. Reusable markdown instructions
3. Executable command integration
4. Skill discovery
5. Skill triggering
6. Skill packaging
7. Skill sharing
8. Team distribution
9. Troubleshooting skill behavior

Subagents

1. Specialized AI subagents
2. Task-specific delegation
3. Isolated context management
4. Main conversation cleanliness
5. Tool restrictions
6. Permission modes
7. Custom prompts
8. Team subagent patterns
9. Subagent governance

Claude Code SDK

1. SDK purpose
2. Programmatic agent workflows
3. Python usage
4. TypeScript usage
5. Query-based interaction
6. Client-based interaction
7. Autonomous file reading
8. Command execution
9. Code editing workflows
10. Agent loop customization
11. Integration into internal tools

8. Model Context Protocol Fundamentals

MCP purpose and positioning

1. Open standard for AI-tool integration
2. Connecting AI applications to external systems
3. Claude and external data sources
4. Claude and external tools
5. Claude and reusable workflows
6. Reducing custom integration burden

MCP architecture

1. MCP clients
2. MCP servers
3. Host applications
4. Transport-agnostic communication
5. Request-response flow
6. Message types
7. Capability negotiation
8. Tool discovery
9. Resource discovery
10. Prompt discovery

MCP core primitives

1. Tools
2. Resources
3. Prompts
4. Model-controlled tools
5. App-controlled resources
6. User-controlled prompts
7. Choosing the right primitive
8. Combining primitives in workflows

Building MCP servers

1. Python SDK setup
2. Server project structure
3. Tool definition
4. Decorator-based tool creation
5. Type hints
6. Field descriptions
7. Input validation
8. Server inspector usage
9. Browser-based testing
10. Debugging server functionality

Building MCP clients

1. Client implementation
2. Server connection
3. Resource reading
4. MIME type handling
5. Prompt access
6. Tool invocation flow
7. External service integration
8. Context injection
9. Autocomplete patterns

MCP resources

1. Read-only data exposure
2. Static URI resources
3. Templated resources
4. Parameterized resources
5. JSON resources
6. Text resources
7. Resource access patterns
8. Resource security boundaries

MCP prompts

1. Pre-crafted instructions
2. Workflow-specific prompts
3. Reusable prompt patterns
4. Prompt discovery
5. Prompt invocation
6. User-controlled workflow guidance
7. Prompt lifecycle management

Advanced MCP considerations

1. Remote MCP servers
2. MCP connector through the Messages API
3. MCP in Claude Code
4. MCP server authentication
5. Tool search with MCP
6. Filesystem access considerations
7. Notifications
8. Sampling concepts
9. Transport mechanisms
10. Production MCP server design

9. Cloud Platform Integration

Claude deployment options

1. Direct Claude API
2. Claude Platform on AWS
3. Amazon Bedrock
4. Google Cloud Vertex AI
5. Microsoft Foundry
6. Platform-operated versus partner-operated access
7. Feature availability differences
8. Billing considerations
9. Compliance considerations

AWS integration concepts

1. Amazon Bedrock access model
2. AWS identity and access management
3. Region availability considerations
4. Model access management
5. Request and response differences
6. Enterprise cloud governance
7. Logging and monitoring
8. Cost management

Google Cloud integration concepts

1. Vertex AI access model
2. Google Cloud IAM
3. Project and region configuration
4. Endpoint usage
5. Service account considerations
6. Enterprise data controls
7. Monitoring and logging
8. Cost management

Azure and managed platform concepts

1. Microsoft Foundry access model
2. OAuth-based access concepts
3. Azure billing integration
4. Enterprise identity alignment
5. Platform feature availability
6. Governance considerations

Choosing an integration path

1. Direct feature access
2. Existing cloud commitments
3. Compliance requirements
4. Procurement constraints

5. Team skill sets
6. Operational ownership
7. Support model selection

10. Security, Privacy, and Governance for Technical Claude Use

Data protection foundations

1. Sensitive data identification
2. Credential protection
3. Secret handling
4. Environment variable safety
5. Personal data handling
6. Confidential document handling
7. Data minimization
8. Retention awareness

Access control

1. API key scope
2. Workspace separation
3. Role-based access
4. Least privilege
5. Key rotation
6. Cloud IAM alignment
7. Tool permission boundaries
8. MCP server trust boundaries

Prompt injection and tool security

1. Prompt injection awareness
2. External content risks
3. Tool output trust boundaries
4. MCP server trust evaluation
5. Web content handling
6. File content handling

7. Command execution safeguards
8. Human approval requirements

Secure coding with Claude

1. Code review responsibility
2. Dependency awareness
3. Secure defaults
4. Generated code validation
5. Test coverage expectations
6. Vulnerability review
7. License awareness
8. Production deployment controls

Governance and operational readiness

1. Acceptable use standards
2. Review and approval workflows
3. Logging and audit trails
4. Monitoring and observability
5. Incident response considerations
6. Change control
7. Documentation requirements
8. Team operating model

11. Practical Technical Adoption

Team workflow design

1. Shared technical prompt patterns
2. Shared Claude Code instructions
3. Repository standards
4. Common command libraries
5. Skill libraries
6. Subagent libraries
7. MCP server catalogues
8. Review responsibilities

Internal enablement

1. Developer onboarding
2. Technical playbooks
3. Prompt and workflow repositories
4. Code review checklists
5. Secure usage guidelines
6. API usage standards
7. Troubleshooting guides
8. Adoption maturity model

Use case selection

1. Coding assistance
2. Internal knowledge assistants
3. Documentation automation
4. API-powered support tools
5. Data processing assistants
6. Workflow automation
7. Engineering productivity workflows
8. Operations support workflows

Implementation planning

1. Feasibility assessment
2. Data readiness
3. Integration readiness
4. Security readiness
5. Cost readiness
6. Pilot planning
7. Success metrics
8. Feedback cycles
9. Scale-up planning

Ongoing improvement

1. Prompt refinement

2. Evaluation updates
3. Tool schema improvement
4. MCP server maintenance
5. Cost optimization
6. Model migration planning
7. Platform update tracking
8. Governance refresh cycles

Disclaimer

This course outline is provided as a general training framework and proposed scope of coverage only. It does not constitute a fixed agenda, binding commitment, official certification pathway, partnership representation, or guaranteed delivery sequence. The trainer reserves the right, at his professional discretion, to amend, expand, condense, reorder, replace, or otherwise modify any portion of the content, emphasis, structure, tools, demonstrations, or delivery approach without prior notice, based on participant background, organizational requirements, technology changes, platform availability, security considerations, and instructional judgment.

© 2026 Faysal Aziz · Kuala Lumpur