Agentic AI Developer Tools Bootcamp

Python Brush-Up · The AI Development Paradigm · Claude Code · Agentic AI · Security & Governance · AI-Integrated APIs & Workflows

5 Days · 4 Hours/Day · 20 Hours Total

Instructor-Led · Hands-on · For .NET engineering teams moving into AI-assisted and agentic development

Format	5 × 4-hour live sessions (virtual or on-site), instructor-led and hands-on.
Audience	Professional .NET developers and tech leads with working proficiency in C#. No prior AI or agentic experience required.
Prerequisites	Comfort with Git, the command line, and REST APIs. No Python experience required — Day 1 builds it from zero.
Tools Covered	Claude Code (primary, in depth), a short orientation to GitHub Copilot and Cursor, MCP (C# SDK v2.0), Microsoft Agent Framework, LangGraph (Python), Open Policy Agent (OPA/Rego).
Practice Material	Two small, generic sample projects: a Python + LangGraph sample agent, and an ASP.NET Core Web API with 3–4 CRUD endpoints and one outbound HTTP call. No knowledge of internal systems is assumed.

Course Overview

This bootcamp takes C#-first .NET engineers from zero Python and no agentic experience to building a small, governed, agent-callable feature on their own. It opens with a Python brush-up, because most of the agentic-AI ecosystem — LangChain, LangGraph, and most vendor sample code — is written in Python. It then contrasts developing with AI against typical .NET Core development, so participants treat this as a different kind of engineering rather than just a different tool.

Claude Code is the deep-dive tool throughout. GitHub Copilot and Cursor get one short, clearly scoped orientation session, so that switching tools later is a syntax change, not a relearn. The second half of the week covers agents in both Python and C#, zero-trust security and AI governance, and exposing APIs to agents through MCP. Every lab runs against two small, disposable sample projects.

Learning Objectives

- Read and lightly modify Python code confidently, and map its constructs to their C# equivalents.
- Explain in concrete terms how developing with AI differs from typical .NET Core development.
- Operate Claude Code fluently — chat, plan mode, auto-accept agent mode, and headless mode — and author custom slash commands and a CLAUDE.md hierarchy.
- Build the same simple agent in Python (LangGraph) and in C# (Microsoft Agent Framework), and explain RAG and the plan → act → observe loop.
- Apply zero-trust principles to AI agents, write a policy-as-code rule in OPA/Rego, and map controls to ISO/IEC 42001 and the NIST AI RMF.
- Expose an API operation as an MCP tool and ship a governed, agent-callable feature with a policy check, an audit log, and a human-approval gate.

Programme Design Choices

- **Python brush-up comes first (Day 1).** This cohort is C#-first, and Day 1 makes sure the Python gap never blocks a later concept.
- **A dedicated mental-model session** (Day 1, Session 3) contrasts developing with AI against typical .NET Core development.
- **Claude Code is the primary tool.** Copilot and Cursor get one short orientation slot (Day 2, Session 4) instead of parallel head-to-head tracks.

Five-Day Arc

Day	Theme	Center of Gravity
1	Python Brush-Up & The AI Development Paradigm Shift	By end of Day 1, “this is a different kind of engineering, not just a different tool” should feel obvious, not abstract.
2	Claude Code as Your Primary Engineering Partner	One tool, deeply — with just enough exposure to the others that switching later is a syntax change, not a relearn.

Day	Theme	Center of Gravity
3	Agentic AI — Concepts, Frameworks & Your First Agent	The same small agent built twice, in the two languages participants will actually use.
4	Security & AI Governance — What's Changing and Why It Matters	Zero-trust for agents is the same instinct as for services — least privilege, defense in depth — aimed at a new kind of caller.
5	Building & Integrating APIs and Workflows with AI	By Friday afternoon, participants have built end to end, on their own, a governed, agent-callable feature.

DAY 1

Python Brush-Up & The AI Development Paradigm Shift

By end of day, participants can:

- Read and lightly modify a small Python script or notebook confidently — functions, classes, dicts, async, pip/venv.
- Explain, in concrete terms, how developing with AI differs from typical .NET Core development.
- Run a first Claude Code session against both a Python project and a C# project.

Time	Session	Focus & Key Concepts	Hands-on Lab
0:00–1:00	Python for .NET Engineers, Part 1: Syntax Mapping	C# class/method ↔ Python class/def; try/except vs. try/catch; pip/venv vs. NuGet/dotnet restore; type hints vs. C# static typing; list/dict vs. List<T>/Dictionary<K,V>.	Take a small, familiar C# class (e.g. a simple validator or calculator) and, with Claude Code, translate it to Python; run both and compare output.
1:00–2:00	Python for .NET Engineers, Part 2: Reading the AI Ecosystem	Decorators, async/await in Python, list/dict comprehensions, f-strings — enough to read, not yet to write from scratch.	With Claude Code narrating line by line, read a short public LangChain or LangGraph "hello agent" quickstart; annotate each unfamiliar construct with its closest C# equivalent.
2:00–3:00	How Developing With AI Differs From Typical .NET Core Development	Deterministic, compiler-checked code vs. probabilistic model output; "the compiler will catch it" vs. "the model might not"; tests as the new specification; the prompt as an interface; where SOLID/DRY still apply and where they change (premature abstraction costs more with AI-generated code, not less).	Given one simple spec, have Claude Code implement it twice — once autonomously end to end, once step by step in plan mode. Compare results and discuss which mode fits which task and risk level.
3:00–4:00	First Contact: Claude Code on a Python Project and a C# Project	CLI install, project init, permission modes (auto-accept / ask-each-time / plan mode); CLAUDE.md basics.	Run **claude init** on the sample Python project and the sample .NET Web API; generate a first CLAUDE.md for each; note how explanations differ between a dynamically typed and a statically typed codebase.

Use case anchor: By end of Day 1, "this is a different kind of engineering, not just a different tool" should feel obvious, not abstract.

Day 1 Assessment & Deliverable

A working Python translation of the Session 1 C# class, committed with a short note on one behavioral difference found · a completed CLAUDE.md for both sample projects.

DAY 2

Claude Code as Your Primary Engineering Partner

By end of day, participants can:

- Operate Claude Code fluently: chat, plan mode, auto-accept agent mode, and headless mode.
- Author a custom slash command and share it as a versioned team asset.
- Build a project-appropriate CLAUDE.md hierarchy.
- Recognize when a teammate using Copilot or Cursor is doing the same job with different mechanics.

Time	Session	Focus & Key Concepts	Hands-on Lab
0:00–1:00	Claude Code Core Mechanics	Chat vs. plan mode vs. auto-accept agent mode vs. headless mode; the permission model underneath all four.	Run the same small feature request through each of the four modes on the sample .NET Web API; compare speed, control, and how much review each result needs before merging.
1:00–2:00	Slash Commands, Built-in and Custom	Built-in commands (/init, /clear, /compact, /review, /permissions); custom commands (.claude/commands/*.md, $ARGUMENTS).	Build one custom command that scaffolds a repeatable task on the sample repo (e.g. “generate a unit test file for this class”); commit it to .claude/commands/ and demo it.
2:00–3:00	Context Engineering	CLAUDE.md hierarchy (project / directory / personal); @-mentions and file references; when to /compact vs. /clear.	Write a CLAUDE.md for the sample .NET Web API encoding two real team conventions (e.g. naming standards and a testing rule); verify the next suggestion respects both.
3:00–4:00	Where Copilot and Cursor Fit — a Short Orientation, Not a Parallel Track	Copilot's inline-completion-first design; Cursor's editor-native Composer; why Claude Code is the primary agentic partner, and how the same ideas (context files, scoped autonomy, custom commands) transfer directly.	Optional, light: run one Copilot inline suggestion or one Cursor Cmd/Ctrl+K edit on the Session 1 task — just enough to see the family resemblance. No head-to-head comparison lab.

Use case anchor: One tool, deeply — with just enough exposure to the others that switching later is a syntax change, not a relearn.

Day 2 Assessment & Deliverable

One custom slash command committed to .claude/commands/ and demoed live.

DAY 3

Agentic AI — Concepts, Frameworks & Your First Agent

By end of day, participants can:

- Explain the plan → act → observe loop and why agent design is a question of scope, not raw autonomy.
- Explain retrieval-augmented generation (RAG) and why grounding reduces hallucination.
- Build and run a first simple agent.
- Know that Microsoft Agent Framework is the current .NET-native path for the same pattern.

Time	Session	Focus & Key Concepts	Hands-on Lab
0:00–1:00	The Agent Loop, From First Principles	Plan → act → observe → repeat; tools; why a narrowly scoped agent beats one giant prompt.	Before writing code, design on paper three narrowly scoped agents for a simple scenario (e.g. a support-ticket triage / diagnosis / fulfilment split) — scope first, code second.
1:00–2:00	RAG & Memory	Embeddings; vector similarity; grounding a model's answer in retrieved data instead of its own recall.	Run a short Python script that embeds three sample documents and retrieves the closest one for a given query.

Time	Session	Focus & Key Concepts	Hands-on Lab
2:00–3:00	Building a First Agent, in Python	LangGraph (or an equally small, well-documented framework); a two-step agent (classify → respond).	Build a two-step agent against the sample Python project with Claude Code assisting; run it against three sample inputs.
3:00–4:00	The .NET-Native Path: Microsoft Agent Framework	Microsoft's 2026 unified successor to Semantic Kernel and AutoGen; agents, type-safe function tools, sequential/handoff orchestration.	Build the same two-step agent in C# with Microsoft Agent Framework on the sample .NET Web API; compare the two implementations side by side.

Use case anchor: You have now built the same small agent twice, in the two languages you'll actually use — that repetition is the whole point of the day.

Day 3 Assessment & Deliverable
A working two-step agent, built once in Python (LangGraph) and once in C# (Microsoft Agent Framework), producing matching output on three sample inputs.

DAY 4

Security & AI Governance — What's Changing and Why It Matters

By end of day, participants can:

- Explain zero-trust principles as applied specifically to AI agents.
- Write and test a simple authorization policy using a policy-as-code tool.
- Explain what a kill-switch and drift detection are, and why enterprises are asking for both in 2026.
- Map a basic control to ISO/IEC 42001 and the NIST AI RMF.

Time	Session	Focus & Key Concepts	Hands-on Lab
0:00–1:00	Why AI Agents Need a Different Security Model	An API key a service holds vs. a credential a probabilistic decision-maker might misuse; least privilege; short-lived, task-scoped credentials; deny-by-default.	Given a scenario (“an agent is asked to delete a customer record”), design on paper the checks that should happen before the action is allowed.
1:00–2:00	Policy as Code: OPA/Rego, Hands-On	Attribute-based access control; the Open Policy Agent CLI; Rego basics.	Install the OPA CLI; write and test a minimal Rego policy that allows one (role, action) combination and denies everything else.
2:00–3:00	AI Governance in Practice	Kill-switches; continuous evaluation and drift detection; audit trails as an architectural requirement, not an afterthought.	Design a simple audit-log schema for “every AI agent action”, and list exactly what a kill-switch must check before honoring a request.
3:00–4:00	Standards Orientation: ISO/IEC 42001 & the NIST AI RMF	What each standard is; the NIST AI RMF's four functions (Govern, Map, Measure, Manage); why enterprise buyers are asking about both in 2026.	Map the Session 3 audit-log design and kill-switch check to the NIST AI RMF function each best supports.

Use case anchor: Zero-trust for agents is the same instinct you already have for services — least privilege, defense in depth — aimed at a new kind of caller.

Day 4 Assessment & Deliverable
A tested Rego policy · an audit-log schema and kill-switch check design, each mapped to a NIST AI RMF function.

DAY 5

Building & Integrating APIs and Workflows with AI

By end of day, participants can:

- Explain how AI agents call external tools and APIs safely (the tool-use pattern).
- Build a minimal MCP server exposing one API operation as an agent-callable tool.
- Explain the trigger → gate → act → record workflow pattern and where human approval belongs in it.
- Complete a short integrated capstone combining the week's skills.

Time	Session	Focus & Key Concepts	Hands-on Lab
0:00–1:00	How Agents Call Tools Safely	The tool-use broker pattern: authenticate the caller, check policy, issue a short-lived credential, execute, log — five steps, always.	Read a short reference implementation of this pattern (provided sample code); diagram its five steps and identify what each one prevents.
1:00–2:00	Exposing an API as an MCP Tool	The Model Context Protocol; the official C# SDK v2.0 (stateless by default, built on ASP.NET Core).	Wrap one endpoint of the sample .NET Web API as an MCP tool; connect Claude Code to it and invoke it live.
2:00–3:00	Workflow Orchestration Patterns	Trigger → gate → act → record; human-approval gates; the long-running-operation pattern (start → event → complete).	Sketch a three-step workflow for a made-up scenario, marking exactly where a human-approval gate belongs and why.
3:00–4:00	Capstone: A Small, Governed, Agent-Callable Feature	Combining the week: an MCP-exposed API operation, a policy check, an audit-log entry, and a human-approval gate on the highest-risk action.	Each participant or pair builds and demos this end to end on the sample repos; the class reviews each demo against the five-step tool-use pattern from Session 1.

Use case anchor: By Friday afternoon, you will have built — end to end, on your own — a governed, agent-callable feature on a real API.

Day 5 Assessment & Deliverable

A demoed, governed, agent-callable feature: one MCP-exposed API operation, one enforced policy check, one audit-log entry, and one human-approval gate — reviewed live against the five-step tool-use pattern.

Appendix A: Quick Reference

Tool	Command	Purpose
Claude Code	/init	Bootstrap a CLAUDE.md for the current project
Claude Code	/clear	Reset conversation context
Claude Code	/compact	Summarize and shrink context to free up the window
Claude Code	/review	Run a structured review of pending changes
Claude Code	/permissions	Inspect or change the current permission mode
Claude Code	Plan mode	Propose changes without executing them — the default for unfamiliar code
Claude Code	Headless mode	Run non-interactively, e.g. inside a CI/CD pipeline
GitHub Copilot (orientation)	/fix, /tests, /explain	Same family of ideas as Claude Code's commands, inline-completion-first
Cursor (orientation)	Cmd/Ctrl+K, Composer	Inline edit and multi-file agentic edit session
Python	python -m venv .venv && pip install -r requirements.txt	Stand up a throwaway environment for the sample project
OPA	opa eval	Evaluate a Rego policy against a sample input from the command line

Tool	Command	Purpose
.NET	dotnet new webapi -n Sample.Api	Scaffold the sample ASP.NET Core Web API used across Days 2–5
.NET	dotnet add package ModelContextProtocol	Add the official MCP C# SDK to the sample Web API

Appendix B: Glossary

Term	Definition
Agent	A bounded unit of AI-driven decision-making with a narrow, documented scope.
Agentic AI	AI that plans and takes multi-step action via tools, not just answers a single question.
Plan-act-observe loop	The core agent cycle: decide the next step, execute it, read the result, repeat.
RAG	Retrieval-Augmented Generation — grounding a model's answer in retrieved data to reduce hallucination.
Vector store / embeddings	A numeric representation of text used for similarity search.
MCP	Model Context Protocol — the open standard for exposing tools and data to AI agents; the official C# SDK v2.0 is stateless-first.
Tool-use broker pattern	Authenticate → authorize → issue short-lived credential → execute → audit, for every tool call.
Zero-trust	No implicit trust for any caller, even inside the perimeter — every call is authenticated and authorized.
OPA / Rego	Open Policy Agent and its policy language; separates authorization logic from application code.
Kill-switch	A mechanism to instantly disable a runaway or misbehaving agent.
Drift detection	Monitoring whether an agent's output quality degrades over time via sampled evaluation.
ISO/IEC 42001	The first international standard for an AI management system.
NIST AI RMF	The U.S. NIST AI Risk Management Framework: Govern, Map, Measure, Manage.
Trigger → gate → act → record	The generic workflow pattern: something starts the work, a checkpoint may pause it, the action executes, the outcome is recorded.
Microsoft Agent Framework	Microsoft's 2026 unified successor to Semantic Kernel and AutoGen for building .NET/Python agents.
CLAUDE.md	Claude Code's per-project (and per-directory / personal) instruction file.
Plan mode	A Claude Code mode that proposes changes without executing them.
Headless mode	Running Claude Code non-interactively, e.g. inside a CI/CD pipeline.

Appendix C: Assessment & Deliverables

Day	Checkpoint(s)
Day 1	A working Python translation of the Session 1 C# class, committed with a short note on one behavioral difference found · a completed CLAUDE.md for both sample projects.
Day 2	One custom slash command committed to .claude/commands/ and demoed live.
Day 3	A working two-step agent, built once in Python (LangGraph) and once in C# (Microsoft Agent Framework), producing matching output on three sample inputs.
Day 4	A tested Rego policy · an audit-log schema and kill-switch check design, each mapped to a NIST AI RMF function.
Day 5 (Capstone)	A demoed, governed, agent-callable feature: one MCP-exposed API operation, one enforced policy check, one audit-log entry, and one human-approval gate — reviewed live against the five-step tool-use pattern.