

IT Roles & Open Source

Reading time: ~10 min · Prereqs: Lessons 01–11 · *Snapshot 2026 — role titles drift, especially around AI engineering; the open-source half is much more stable.*

Why this matters to you

You've now seen most of the surface area of modern IT. The last question is the one this course was secretly building toward all along: *where in IT do you actually want to land?* This lesson is two halves that answer that. The first half is **roles** — who actually does what, in what kind of team, with what kind of background; the map of jobs that hangs over the map of technology you've been building since **Lesson 01 (the IT map)**. The second half is **open source** — the cultural and economic substrate underneath almost every job, and the most common on-ramp into a paying IT career. And then, at the very end of this lesson and of this whole course, an explicit recommendation: *which course on this same learning platform you should take next*, based on which lessons in this course made your eyebrows go up.

1. Roles are clouds, not boxes

The job titles you see on LinkedIn — "Frontend Engineer", "Backend Engineer", "DevOps Engineer", "Site Reliability Engineer", "ML Engineer", "Platform Engineer" — sound like crisp categories. Inside any real team, they aren't. At a five-person startup, one engineer does the work that, at a 5,000-person company, would be split across fifteen specialized titles. The same person carries different job titles at different points in their career without actually changing what they do day to day. The technologies you saw in **Lesson 05 (the tools)** — IDEs, GitHub, AI assistants, terminals — sit underneath all of these titles equally. So when you read this lesson's role names, hold them loosely. **Use them to know what to search for and what conversations to have**, not to box yourself in. The cluster matters more than the exact noun on the business card.

2. The engineering job families

The biggest cluster is **software engineering: frontend** (what runs in the browser — React, Vue, TypeScript), **backend** (what runs in the data center — Python/Django, Node, Go, Java/Spring), **full-stack** (both), **mobile** (iOS / Android), **embedded** (firmware in your car, your headphones, your fridge), and **game** (Unity, Unreal). This is the cluster most people picture when they hear "programmer".

The second cluster is **infrastructure and operations: DevOps Engineer, Site Reliability Engineer (SRE), Platform Engineer, Cloud Engineer** — the people who keep the software actually running, deploy new code safely, own the on-call rotation, and build the internal developer-tooling that the first cluster ships on top of. The boundary between "DevOps" and "SRE" and "Platform" shifted noticeably in the late 2020s; *as of 2026*, treat them as three names for overlapping work, with SRE being the most reliability-and-error-budget focused, and Platform being the most internal-developer-tooling focused. The third cluster — security — is now substantial enough to deserve its own section.

3. Data and AI roles (as of 2026)

Five roles, with shifting boundaries:

- **Data Engineer** — builds the pipes (Airflow, dbt, Spark, Kafka) that move data from where it's produced to where it's queried.
- **Data Analyst** — asks questions of the data (SQL, dashboards in Looker / Tableau / Metabase), often embedded in a business team.
- **Data Scientist** — builds statistical and ML models on top of the cleaned data, usually in notebooks.
- **ML Engineer** — takes a data scientist's notebook and turns it into a production service.
- **AI Engineer** — a newer title (broadly adopted starting in 2023) for someone who builds *with* LLMs, RAG, and agents — closer to a software engineer with deep prompt and orchestration craft than to a classical ML researcher. This is the role most directly born of **Lesson 13 (the LLM & agent era)**.

Boundaries here moved meaningfully between 2022 and 2026 as LLMs reframed what "doing ML" means in industry. Treat any role-list this specific as a 12-month snapshot, not a permanent taxonomy.

4. Security as its own world

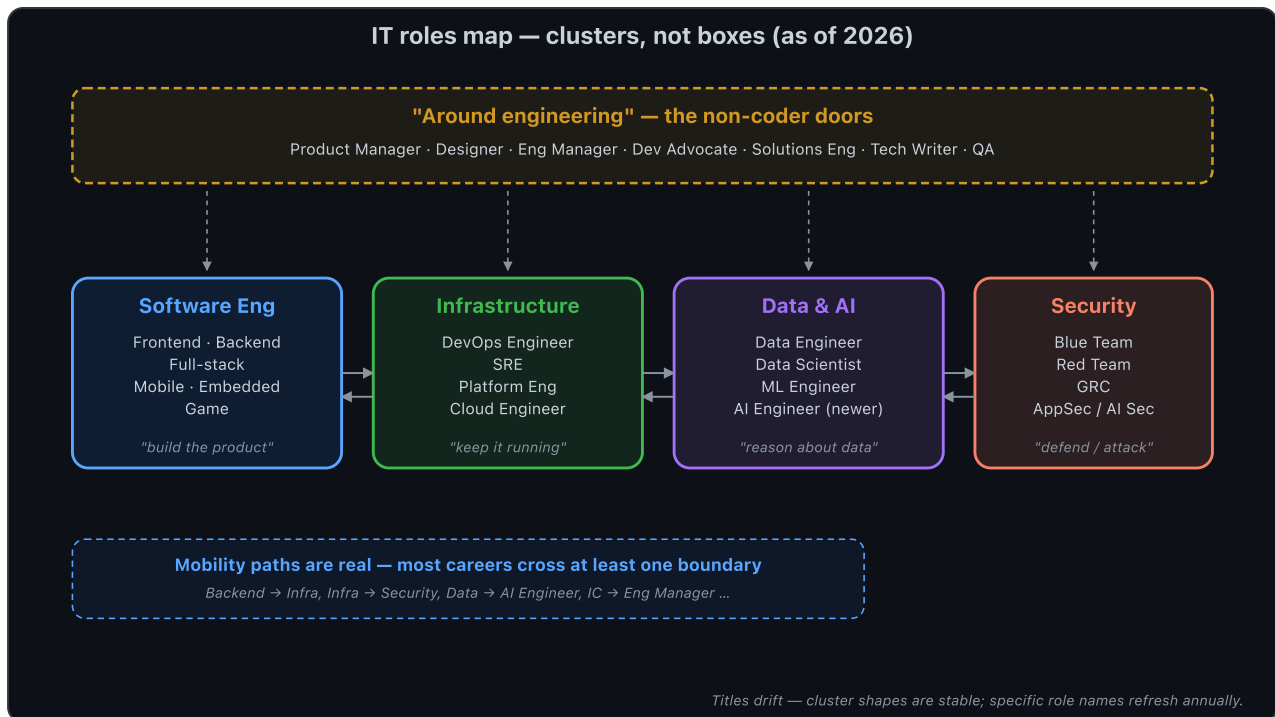
Defensive (Blue Team) — SOC analysts, threat hunters, incident responders, detection engineers; the people watching the logs at 3 AM and writing the playbooks. **Offensive (Red Team)** — penetration testers, security researchers, exploit developers; the people *being* the simulated attacker so the defenders can practice. **Governance, Risk, Compliance (GRC)** — the audits, the policies, the SOC 2 reports, the regulator-facing work; often the door into security from a non-engineering background. And then specializations: **AppSec** (security of the application code), **Cloud Security**, **OT / ICS Security** (industrial systems), **AI Security** (model, prompt, supply-chain — born of the world introduced in Lesson 13). Security is now its own career ladder, not just a senior-engineer rotation.

5. The non-coder doors into IT

Many of the highest-paid and most-influential people in IT do not write production code. **Product Manager** — what to build, for whom, why; sits between engineering and the rest of the business. **Designer** — UX, UI, research; the human face of the product. **Engineering Manager** — people + delivery; "1:1s and unblock people" by the hour. **Solutions / Sales Engineer** — technical, customer-facing, helps prospects evaluate the product. **Developer Advocate** — community, content, conferences; reaches developers where they are. **Technical Writer** — the documentation, the API reference, the changelog; rare and very valued. **QA / Test Engineer** — designs the way the product gets stressed. These doors don't require a computer-science degree, but they do require the vocabulary you've built across this course.

6. The roles map, one diagram

Here is the same idea drawn as clusters arranged spatially. The thin lines suggest common career-mobility paths — people move between adjacent clusters more often than the org chart implies.



7. Open source — what it actually is, and why it works

Open-source software is code anyone can read, copy, and change — usually for free. Almost every program you run, every website you visit, every cloud service you use, sits on a tower of open-source pieces underneath. Linux runs most of the internet. Python, JavaScript, and Go are open. The web browser you're reading this in is largely open. Postgres, Redis, Kubernetes, Git itself — all open.

Why does it work, when nobody is paid by you for it? Three reasons. **Dogfooding** — the maintainers needed the tool too, and built it for themselves; you benefit as a side effect. **Corporate backing** — most "open source" you depend on (the Linux kernel, Kubernetes, Postgres, the V8 JavaScript engine, the LLVM compiler, Python itself) is overwhelmingly maintained by engineers paid by large companies who depend on those projects. **Foundations** — the Linux Foundation, Apache, CNCF, the Python Software Foundation, Mozilla act as legal-and-fundraising homes for projects too important to belong to any one vendor. Licenses matter too: **MIT** and **Apache 2.0** let you do almost anything (including build a closed-source product on top); **GPL** requires you to share your changes back if you redistribute. Most new projects *as of 2026* choose MIT or Apache.

What people argue about

- **Is "open source" being captured by VC-backed companies?** Several influential projects (HashiCorp's stack, Redis, MongoDB, Elasticsearch) shifted their licenses from true open source to "source-available" (BSL, SSPL) in the past five years, to prevent cloud providers from monetizing the project without contributing back. Critics call this a betrayal of the original promise; defenders call it pragmatic survival. The fight is far from settled.
- **Will AI assistants reduce the number of engineers needed?** Three years into the LLM era, the evidence is "engineers are faster, but headcount is not falling and ambition per engineer is rising". Predictions for the next 3–5 years range across the full spectrum. Pick a camp at your peril.
- **CS degree vs bootcamp vs self-taught?** All three still produce hireable engineers in 2026. Increasingly the deciding factor is what you've shipped in public, not which door you came through.

Reading on — the course you take next, on this same platform

This section is unlike any other "Reading on" in this course. Instead of pointing outward, it points sideways — at the other courses sitting in this same repo. Pick one based on which lesson in this course made your eyebrows go up.

- **If Module 4 (the AI era) is the part that made you lean in:** take the **Claude & LLMs** course (33 lessons — the longest course on this platform; the depth answer to Lesson 13). Then, if you want to build tools that LLMs use, take **MCP Servers** (focused on the Model Context Protocol — how Claude and other LLMs talk to your tools). If you want to build LLM-driven agents end-to-end, take **Building Your Own Agents**.
- **If Module 3 (security) was your favorite:** take **CTF Blue Team** (defender exercises in a Dockerized lab) or **CTF Red Team** (offensive techniques in the same kind of lab). For AI-specific security — model, prompt, supply-chain — take **AI OpSec** and **AI Prompt Security**. For password-cracking and crypto practice, take **Hashcat** (10 lessons on practical password cracking).
- **If Module 1 + Module 3 (infrastructure, networking, deployment) felt at home:** take **Linux Networking** (deep dive on routing, namespaces, policy-based routing)

and **Helm Charts** (20 lessons on Kubernetes packaging — the next step beyond the cloud and DevOps lessons of this course).

- **External, for context, not next-step: Roadmap.sh** publishes role-specific learning paths (frontend, backend, DevOps, AI engineer) that go alongside whichever sibling course you choose; **The Pragmatic Engineer** newsletter (Gergely Orosz) is the best read on the industry-trend side — salaries, layoffs, hiring climate, what the senior engineers are actually arguing about.

The course in three sentences

You now have a **map of IT** (Lesson 01) — frontend, backend, data, infra, security all on one page in your head. You've seen **what the tools actually look like** (Lesson 05) — IDEs, GitHub, AI assistants — and you know the words on a job description aren't magic. And you've watched **AI become a real seat at the table** (Lesson 13) — and you know which sibling course on this platform to take to actually build with it. This was a survey course; the map is in your head now. The next step is to go deep on one square of it. Pick a course from the list above. Or go talk to someone who works in IT — you can keep up now.

LAST-REVIEWED: 2026-05-26