TECH RECRUITMENT 101

Recruiter Credibility with Engineers

How I build trust without pretending to be technical

PART OF THE *TECH RECRUITMENT 101* SERIES

Recruiter Credibility with Engineers

How I build trust without pretending to be technical

CONTENTS

Recruiter Credibility with Engineers: How I build trust without pretending to be technical

I wrote this because I kept seeing the same problem. Recruiters were told to build credibility with engineers by sounding more technical. So they borrowed jargon, copied stacks into outreach, and tried to smooth over gaps with confident language. It rarely worked. After building recruiting teams a few times, I learned that engineers do not expect us to be engineers. They expect us to be accurate.

That matters even more now. Candidates are more skeptical of polished language, more wary of automation, and less willing to spend time on a process that feels careless. Hiring managers feel that pressure too. They need recruiters who can clarify a fuzzy brief, run a useful first conversation, and bring back notes that help them decide. That is what prompted this book.

So I focused on the places where trust is either built or lost. You will see how credibility starts before the first call, with a better intake and a clearer map of the role. You will see how to talk to engineers without bluffing, how to ask questions that reveal ownership and scope, and how to turn messy interview feedback into something a hiring manager can use. I also cover a short recap I come back to every time, and what changes when AI enters the workflow but trust still depends on human judgment.

This book is part of the Tech Recruitment 101 series because these are foundational skills, even if they do not always get taught that way. If you already know the basics of recruiting, this will help you tighten the parts that matter most: precision, preparation, honesty about what you know, and follow-through when details start to slip.

Why engineers trust some recruiters and ignore others

Most recruiters think credibility with engineers comes from sounding technical. In my experience, that is where trust often breaks. Engineers do not need you to explain distributed systems or debate frameworks. They need you to be correct, clear, and useful.

Many engineers start from skepticism, not enthusiasm. In the Stack Overflow 2025 Developer Survey (https://survey.stackoverflow.co/2025/), 46% said they distrust the accuracy of AI tools, while 33% said they trust them. That skepticism shapes how they read recruiter messages too. If your note sounds polished but vague, they assume the substance may be shaky.

They are also not waiting by the inbox. That does not mean they are rude. It means your message has to earn attention fast.

Engineers usually spot bluffing in small ways. You call a backend role "full stack" because the company may add frontend later. You say "strong in AI" when the team really wants someone who can use existing tools well, not build models. You describe an SRE (site reliability engineer, usually focused on keeping systems stable and available in production) as a DevOps person (usually focused on improving how software is built, deployed, and run) "who does cloud stuff." None of this sounds dramatic. All of it signals that the recruiter is filling gaps with guesses.

False credibility is performative. It leans on buzzwords, borrowed jargon, and long lists that no real team wrote. Real credibility is operational. It shows up when you ask what the team is building, what this person will own early, and which skills are required on day one versus learned after joining.

A strong recruiter reduces work for engineers. A weak one creates more of it. If a candidate has to spend fifteen minutes correcting the role, untangling the interview process, or figuring out whether the salary is real, trust drops. If a hiring manager has to rewrite your notes after every screen, trust drops there too.

The same principle applies when AI enters the process. LinkedIn's Future of Recruiting 2025 (https://business.linkedin.com/content/dam/me/business/en-us/talent-solutions/resources/pdfs/future-of-recruiting-2025.pdf) argues that recruiting teams need both AI capability and human judgment. That is right. AI can help you summarize notes or spot patterns. It cannot rescue a vague brief or verify claims it invents with great confidence and terrible manners.

Candidates notice when automation has replaced attention. In Dice's The Trust Gap in Tech Hiring 2025 (https://www.dice.com/technologists/ebooks/trust-gap-in-the-tech-job-market/), 71% of tech professionals said AI is weakening trust in hiring, and 56% think no human ever sees their résumé. If your process feels generic, hidden, or keyword-driven, candidates assume they are being filtered rather than understood.

False vs real credibility

FALSE	REAL
Uses jargon to sound informed	Uses plain language and gets facts right
Sends broad outreach	Explains why the role fits
Repeats buzzwords from the brief	Tests the brief for gaps and contradictions
Pretends to know	Says what they know and what they will confirm
Hands over messy notes	Brings clear, decision-ready summaries

Engineers are not judging whether you can code. They are judging whether working with you will save time or waste it. They ask, often silently: Does this person understand the shape of the problem? Can they carry details without dropping them? Will they follow through?

Trust is rarely won in one impressive conversation. It builds in small moments. You send the right salary range the first time. You explain why the team opened the role. You capture that the

candidate led migration work, but has not managed people. You come back when you said you would.

If you want engineers to trust you, stop trying to sound like an engineer. Start behaving like a reliable operator. Accuracy beats fluency. Clarity beats polish. Honest limits beat confident nonsense.

Get credible before the first conversation

Credibility starts before the call. If I sound clear in an intake meeting or a first candidate conversation, it is usually because I did the quiet work first.

Most weak recruiter conversations fail before anyone joins Zoom. The brief is vague. The title is inflated. The requirements mix real needs with wishlist items. Then the recruiter tries to compensate with jargon. Engineers spot that fast.

Your job is not to become an engineer overnight. Your job is to build a working map of the role. That map should tell you what this person will do, what they must know on day one, what they can learn after joining, and what problem is serious enough to justify hiring.

I start with the business problem, not the stack. “We need a senior backend engineer” is not a problem. “Our payment service breaks under peak load and nobody owns the reliability work” is a problem. The second version tells you what matters and why the role exists.

That shift matters because titles drift. A Platform Engineer in one company may look like DevOps. In another, the same title may sit closer to SRE. In plain terms, platform engineering usually means building internal tools and infrastructure other engineers use, DevOps usually means improving how software is built, deployed,

and operated, and SRE usually means keeping systems reliable in live production environments. If you only memorize title-to-keyword matches, you will misread the role.

A good intake helps you separate four things: mission, must-haves, learnable skills, and fiction. Mission is the outcome the hire needs to drive. Must-haves are the skills without which the person cannot function early. Learnable skills are useful but trainable. Fiction is manager wishful thinking dressed up in bullet points.

My pre-intake prep

- ✓ Rewrite the brief in plain English
- ✓ Mark each requirement: must, learn, or wish
- ✓ List the team problem this hire solves
- ✓ Compare title with 2 adjacent titles
- ✓ Draft 5 clarifying questions
- ✓ Flag vague words: strong, expert, culture fit

Rewrite the job description before the meeting. If the original says, “drive scalable cloud-native architecture across cross-functional stakeholders,” turn it into plain language. If you cannot translate the sentence, you do not understand it yet.

Then mark each requirement. I label them in three buckets: must have, can learn, and nice to have. Most job descriptions read as if the manager tried to order one human and three backups.

Pressure-test the brief with questions that reduce ambiguity. “When you say senior, what decisions will this person make alone in the first month?” is useful. “Which part of the stack is hardest for the current team to hire?” is useful. “If we find someone great without Kubernetes, would that be a no, or something they could pick up?” is useful.

Listen for evidence, not adjectives. “Strong communicator” means nothing until the manager explains the work. Do they need someone to write design docs, handle incident updates, or push back on product scope? Once you hear the behavior, you can assess for it.

The same rule applies to technical terms. Do not collect acronyms like trading cards. Build enough context to place them. If the role mentions APIs (application programming interfaces, the ways software systems connect and exchange data), know whether the engineer will design them, maintain them, or integrate with

them. If it mentions machine learning, know whether they need someone building models, deploying them, or using off-the-shelf tools safely.

AI can help with the first draft of this prep, but not the final judgment. You can use it to summarize a long job spec, compare nearby titles, or suggest intake questions. But if the brief is weak, AI often turns confusion into polished confusion.

By the time you speak to a candidate, you should be able to answer four clean questions: Why does this role exist? What does success look like in the first six to twelve months? What is truly required at the start? What would make a good candidate stronger but not disqualify them if missing? If you can answer those without hiding behind buzzwords, you will sound credible from the first conversation.

Talk to engineers in a way that feels useful

Once the prep is solid, the conversation gets simpler. A credible recruiter call is not a watered-down technical interview. It is a clean exchange. The candidate should feel understood. You should leave with accurate signal.

Start by lowering the friction. Tell the candidate what you want to cover, how long you need, and what you can answer. “I’d like to give you a clear picture of the role, understand the kinds of systems you’ve worked on, and see whether this is worth a deeper conversation.” Most engineers relax when they know they are not walking into a trivia quiz disguised as a recruiter screen.

Do not open with a stack dump. “We need Go, Kubernetes, Kafka, AWS, Terraform, microservices, CI/CD” tells the candidate almost nothing. Even if those are the tools, lead with the work. Kafka is a tool for moving data between systems, and CI/CD means continuous integration and continuous delivery, the process teams use to test and ship code more safely and often. Describe the work first. What does the team build? Who uses it? What breaks if they get it wrong? Why is the role open?

Good role framing is concrete. “This team owns the internal platform other product teams use to ship services. Right now they need someone who can improve reliability and reduce painful deployments.” That gives a candidate something to react to.

Ask questions that pull out specifics. Yes-no questions kill signal. “Do you have microservices experience?” invites a thin answer and rewards keyword matching. Ask instead, “What kinds of systems have you worked on?” Then narrow. “Where did complexity live?” “What did you own directly?” “What changed because of your work?”

Precision matters more than depth. You do not need to explain distributed systems better than the candidate. You need to capture the truth correctly. There is a big difference between “designed the service boundaries,” meaning they decided how a larger system should be split into separate parts, “maintained a set of services in production,” meaning they kept live customer-facing systems running reliably, and “deployed a service once as part of a larger team,” meaning they contributed to a launch without owning the system end to end. Those describe very different levels of ownership and usually very different seniority. Hiring teams forgive a recruiter who asks a clean follow-up. They do not forgive notes that flatten those into the same bullet.

Use the candidate’s language, but do not parrot it. If someone says, “I worked on event-driven systems with Kafka,” you can mirror that with care: “Got it. Event-driven architecture, Kafka, and your piece was the consumer service for billing events.” That shows you listened and tests your understanding.

When you do not know a term, admit it fast and ask for the useful version. “I know the label, but I want to make sure I capture your part accurately. What did that mean in practice on your team?” You lose more authority by pretending to understand and then writing nonsense.

Visual block error: Expected ',' or '}' after property value in JSON at position 432 (line 1 column 433)

Listen for verbs. Built, migrated, debugged, scaled, reduced, led, supported. Verbs tell you what happened. Nouns alone do not. “Worked with Kubernetes” is weak. “Moved services to Kubernetes and handled rollout failures” is useful.

Bad recruiter calls usually fail in one of two ways: bluffing or passivity. The first sounds technical and falls apart the moment the candidate answers seriously. The second is so afraid of getting it wrong that it learns nothing beyond titles and tools.

One rule helps junior recruiters immediately: if you cannot explain why you asked a question, do not ask it. Every question should help you understand scope, ownership, complexity, collaboration, or motivation.

Become the hiring manager’s most reliable translator

Your credibility does not stop with candidates. It rises or falls inside the hiring process. Hiring managers notice when your summaries make the next decision easier.

The job is not to sound technical. The job is to reduce noise. That means you bring back evidence, not fog. You help the team compare candidates against the same target. You catch drift before the search turns into a scavenger hunt.

A good recruiter summary answers three questions fast: What did we learn? What is still unclear? What should we do next? If a hiring manager has to read five paragraphs to find the signal, you did not save them time.

Use notes that separate facts from interpretation. “Built a payment service used by three teams” is evidence. “Strong backend profile” is a conclusion. Conclusions matter, but they should sit on top of evidence, not replace it.

That distinction matters in debriefs too. One interviewer says, “I did not love the communication.” Another says, “Not senior enough.” Those comments may be valid. They may also mean nothing until someone asks for examples. What did the candidate

say? Which answer showed weak scope, weak clarity, or weak judgment? If nobody can point to behavior, the team is grading vibes.

Turn vague feedback into usable feedback

VAGUE	USEFUL
Not technical enough	Could explain API tradeoffs, like the pros and cons of different ways software systems connect, but struggled on database indexing, meaning how data is organized so queries run quickly
Poor communication	Answered clearly one-to-one, but lost structure in system design
Not senior	Solved the problem, but did not anticipate rollout risks or team impact

You do not need to override interviewers. You need to ask clean follow-up questions. “What did you hear that led you there?” works better than “Are you sure?”

You also need to notice when the target moves mid-search. It happens all the time. A team opens a role for a backend engineer, then falls in love with platform experience, then starts asking for

data depth, then wonders why the pipeline is thin. When feedback keeps changing, your problem is not sourcing. Your problem is calibration.

Spot the pattern early. If rejected candidates are being rejected for different reasons, review the must-haves again. Ask the hiring manager to separate true requirements from nice-to-haves. A search can survive a hard market. It usually cannot survive a blurry target.

This is one reason structured scorecards help. They force the team to assess the same areas in the same language. They also make disagreement useful.

AI tools can help here, but they are support tools, not judgment tools. They can capture transcripts, pull themes, and draft scorecard notes. But if your notes read like a polished transcript with no judgment, people will feel it.

Hiring managers also trust you when you make trade-offs visible. Very few candidates are perfect. A useful summary sounds like this: “Strong distributed systems experience, weaker mentoring examples, likely strong for this team because the immediate gap is architecture.” That helps a manager choose. “Overall promising candidate” does not.

Keep your written updates short enough to skim and sharp enough to act on. Engineering leaders are overloaded with communication already. In the LeadDev Engineering Leadership

Report 2025 (https://leaddev.com/wp-content/uploads/2025/07/LeadDev-Engineering-Leadership-Report-2025.pdf), 58% said they were spending more time communicating with teams, customers, and stakeholders than the year before. If you can turn interview noise into a decision-ready brief, you become useful very fast.

The trust-building checklist I come back to every time

A short recap:

Trust-building checklist

- ✓ Know the real problem behind the role
- ✓ Separate must-haves from nice-to-haves
- ✓ Write outreach that proves relevance
- ✓ Never fake understanding
- ✓ Check notes for technical accuracy
- ✓ Close loops quickly
- ✓ Bring back evidence, not adjectives
- ✓ Flag process confusion early

Know the real problem behind the role. Before you source a single person, ask why this hire exists. Is the team replacing someone, adding capacity, or trying to fix a system that keeps breaking?

Separate must-haves from nice-to-haves. Hiring managers often hand over a wish list dressed as a job description. Your job is to turn it into a decision tool.

Write outreach that proves relevance. Good outreach shows why this person, for this role, makes sense. Mention one real connection: their work with distributed systems, their move into data platforms, their experience leading migrations.

Never fake understanding. Engineers can forgive a gap. They rarely forgive bluffing.

Check your notes for technical accuracy. This sounds small until a hiring manager sees that you wrote Python when the candidate said Go, or platform work when they described developer tools.

Close loops quickly. Silence makes people assume the process is messy or careless. If you do not have a decision, send an update anyway. Gartner found only 26% of applicants trust AI to evaluate them fairly (https://www.gartner.com/en/newsroom/press-releases/2025-07-31-gartner-survey-shows-just-26-percent-of-job-applicants-trust-ai-will-fairly-evaluate-them). When candidates already suspect a black box, simple human follow-through matters more.

Bring back evidence, not adjectives. “Strong candidate” tells a manager nothing. “Led a cloud migration, reduced incident load, and wants hands-on architecture work” gives them something to evaluate.

Flag process confusion early. If the interview loop tests one thing while the hiring manager says they care about another, raise it before candidates enter the funnel. In HackerRank's 2025 Developer Skills Report (https://pages.hackerrank.com/hubfs/PDFs/HackerRank%202025%20Developer%20Skills%20Report.pdf), 78% of developers said assessments do not match real work.

What credibility looks like in an AI-shaped hiring market

AI has changed the surface of technical recruiting. It has not changed the job underneath. We still earn trust the same way: by being precise, honest, and useful.

The market is noisier now, especially around AI-flavored roles. A software engineer role may need some AI literacy without being a machine learning role. A machine learning engineer may build production systems, not research models. An MLOps engineer may sit closer to platform engineering than data science. In simple terms, MLOps means building and running the systems that help machine learning models get deployed, monitored, and updated reliably. Platform engineering usually means building internal tools and infrastructure for other engineers, while data science usually focuses more on analyzing data and building models. Your job is not to fake depth. Your job is to clarify what the team actually needs.

The fastest way to lose credibility is to treat every AI-adjacent role as magic. Ask simple questions that force definition. Does this person need to train models, fine-tune existing ones, evaluate outputs, build data pipelines, or ship product features that call an API? Those are different jobs. Training models usually means creating a model from data. Fine-tuning means adapting an existing model to perform better on a specific task. Evaluating outputs means checking whether the model's answers are useful

and safe. Building data pipelines means moving and preparing data so systems can use it reliably. Shipping product features that call an API means adding user-facing features that send requests to another software service.

Candidates notice sloppy language fast. In the Stack Overflow 2025 Developer Survey (https://survey.stackoverflow.co/2025/ai%23developer-tools), 46% said they distrust the accuracy of AI tools, while 33% said they trust them. So if your outreach reads like polished nonsense, they will assume the rest of the process works the same way.

Automation creates a second trust problem. Many candidates feel screened by systems they cannot see and do not trust. Gartner found only 26% of applicants trust AI to evaluate them fairly, while 52% believe AI is screening their application information in some way, according to Gartner's 2025 survey (https://www.gartner.com/en/newsroom/press-releases/2025-07-31-gartner-survey-shows-just-26-percent-of-job-applicants-trust-ai-will-fairly-evaluate-them).

Use AI to speed up the parts that do not require judgment. Clean up notes. Draft outreach. Summarize patterns. Then check every output that reaches a candidate or a hiring manager. If a message includes the wrong stack, a lazy claim, or a vague salary range, the tool did not make that mistake. We did.

Credibility rules for AI-era recruiting

- ✓ Define the work before the title
- ✓ Explain where automation is used
- ✓ Review every candidate-facing message
- ✓ Write notes as evidence, not impressions
- ✓ Escalate technical ambiguity to the manager

AI can make you faster, but it cannot make you credible. Credibility still comes from clean judgment, clear language, and a process another human can trust. LinkedIn's Future of Recruiting 2025 (https://business.linkedin.com/content/dam/me/business/en-us/talent-solutions/resources/pdfs/future-of-recruiting-2025.pdf) makes the same point in a more polite way.

The thread through all of this is simple. Do the prep before the call. Ask questions that uncover ownership, not just keywords. Turn candidate conversations into evidence. Push vague feedback until it becomes usable. And when AI speeds something up, check that it did not also flatten the truth.

If you want a practical next step, do not try to overhaul your whole process at once. Take the trust-building checklist from this book and use it on your next intake, your next recruiter screen, and your next candidate summary. Then look for the small places where trust usually leaks: inflated titles, fuzzy must-haves, weak notes, slow follow-up. Fix those first.

That is how credibility grows. Not through performance. Through consistency. If this was useful, the other books in the Tech Recruitment 101 series will help you build the same kind of clarity in the rest of your process, one part at a time.

Glossary

- **AI (Artificial Intelligence)** — Technology used to perform tasks that normally require human judgment, such as summarizing notes or analyzing patterns. In recruiting, it often appears in sourcing, screening, outreach, and workflow tools.
- **AI-adjacent role** — A role that touches AI work without necessarily being a core AI or machine learning position. These jobs may involve using AI tools, integrating AI features, or supporting AI systems.
- **AI capability** — The ability to use AI tools effectively in a work process. In this book, it refers to recruiting teams using AI while still relying on human judgment.
- **API (Application Programming Interface)** — A way for different software systems to connect and exchange data. Recruiters should know whether a role involves designing, maintaining, or using APIs.
- **API tradeoffs** — The pros and cons of different ways software systems can connect and communicate. This usually relates to decisions about usability, performance, flexibility, and reliability.
- **Application information** — The data a candidate submits as part of a job application, such as résumé details or answers in a form. In this context, it may be screened by AI systems.
- **Architecture** — The high-level design of a software system and how its parts fit together. In hiring, it often signals

ownership of technical decisions rather than just coding.

- **Assessments** — Tests or exercises used to evaluate candidates' skills during hiring. In technical hiring, these may include coding, system design, or job-simulation tasks.
- **AWS (Amazon Web Services)** — A popular cloud platform used to run software, infrastructure, and services online. Recruiters often see it listed as part of a team's stack.
- **Backend engineer** — An engineer who works on the server-side parts of software, such as business logic, databases, and system integrations. They usually build the systems users do not see directly.
- **Billing events** — System-generated messages related to billing actions, such as charges or payments. In event-driven systems, these events trigger other services to take action.
- **CI/CD (Continuous Integration / Continuous Delivery)** — Automated processes teams use to test and ship code more safely and more often. It helps software move from development into production faster and with less manual effort.
- **Cloud** — Internet-based computing infrastructure used to run software, store data, and scale systems. In hiring, "cloud" can refer to platforms like AWS and the operational skills needed to manage systems there.
- **Cloud migration** — Moving systems, applications, or data from one environment into the cloud. This often involves technical planning, risk management, and production changes.

- **Cloud-native architecture** — A way of designing software so it works well in cloud environments, often emphasizing scalability, resilience, and distributed services. The phrase is often overused, so recruiters should ask what it means in practice for the team.
- **Code** — The written instructions developers create to build software. Engineers test, change, and deploy code as part of their day-to-day work.
- **Consumer service** — A service in an event-driven system that receives and processes messages or events from another system. It is one part of how systems communicate asynchronously.
- **Cross-functional stakeholders** — People from different teams or functions who are involved in a project, such as engineering, product, or operations. In technical roles, this often signals collaboration beyond pure coding.
- **Data pipelines** — Systems that move, transform, and prepare data so it can be used reliably by applications or models. They are common in data, analytics, and machine learning environments.
- **Data platforms** — Shared systems and infrastructure used to store, process, and serve data across a company. Engineers working on them often focus on scale, reliability, and internal enablement.
- **Data science** — Work focused on analyzing data and building models to generate insights or predictions. It is different from

platform or infrastructure work, even when both support AI efforts.

- **Database indexing** — A way of organizing data so queries run faster. It is a common technical concept that can indicate database performance knowledge.
- **Debugged** — Found and fixed problems in software or systems. In candidate conversations, this verb helps show hands-on troubleshooting ownership.
- **Deploy / Deployed / Deployments** — The act of releasing software into a live or test environment where it can run. Recruiters should distinguish between contributing to a deployment and owning the deployment process.
- **Design docs** — Written documents that explain how a technical system or solution should be designed. They are often used to communicate decisions and align engineering teams.
- **Developer tools** — Software tools built to help engineers write, test, deploy, or manage software more effectively. These can be internal platform tools or external products.
- **DevOps** — A practice and role area focused on improving how software is built, deployed, and run. In some companies it overlaps with platform engineering or infrastructure work.
- **DevOps person** — Informal shorthand for someone working in DevOps-related areas. In this book, it is used as an example of an oversimplified and potentially inaccurate label.
- **Distributed systems** — Software systems made up of multiple connected parts running across different machines or

services. They are common in modern backend, platform, and infrastructure environments.

- **Engineering leaders** — Senior engineering decision-makers, such as heads of engineering or engineering managers, who help set hiring direction and evaluate candidates. They are often overloaded and need clear recruiter summaries.
- **Engineering teams** — Groups of engineers working together to build and maintain software systems. Recruiters often partner with them to understand role requirements and interview feedback.
- **Event-driven architecture** — A system design approach where actions are triggered by events or messages rather than direct step-by-step calls. It is common in systems that need to handle many independent processes.
- **Event-driven systems** — Systems built around events that signal something happened, prompting other services to respond. They are often used in scalable and decoupled software designs.
- **Fine-tune / Fine-tuning** — Adapting an existing machine learning model to perform better on a specific task or dataset. It differs from training a model from scratch.
- **Frameworks** — Reusable software structures and tools developers use to build applications more efficiently. Engineers may care about them, but recruiters should focus on how they are used in the role.
- **Frontend** — The user-facing part of software, such as the interface people interact with in a browser or app. It contrasts

with backend work.

- **Full stack** — A role or skill set covering both frontend and backend development. The term can be used too loosely, so recruiters should confirm what mix of work the team actually needs.
- **Go** — A programming language often used for backend, infrastructure, and performance-oriented systems. It is sometimes listed in a team's technical stack.
- **Infrastructure** — The underlying systems, services, and resources that software runs on, such as servers, networks, and cloud platforms. Platform, DevOps, and SRE roles often work closely with infrastructure.
- **Integrate / Integrate with** — To connect one system, tool, or service with another so they work together. In technical roles, this can mean using APIs or linking systems in production.
- **Internal platform** — Shared internal systems and tools that other product or engineering teams use to build and ship software. Teams owning an internal platform usually focus on enablement, reliability, and developer efficiency.
- **Internal tools** — Software built for employees rather than external customers. In engineering, these often help other developers work more efficiently.
- **Interview loop** — The sequence of interviews and assessments a candidate goes through during a hiring process. In technical hiring, it often includes recruiter, manager, and technical interview stages.

- **Kafka** — A tool used to move data and messages between systems. It commonly appears in event-driven architectures and high-scale data workflows.
- **Kubernetes** — A system for running and managing containerized applications in production. It is commonly associated with infrastructure, platform, and cloud-native environments.
- **Machine learning** — A field of AI focused on training systems to learn patterns from data and make predictions or decisions. Roles in this space can vary widely between research, product, and infrastructure work.
- **Machine learning engineer** — An engineer who works on building, deploying, or scaling machine learning systems in production. This role often blends software engineering with model-related work.
- **Microservices** — An approach to building software as multiple smaller services rather than one large application. Recruiters should ask what the candidate actually owned within that setup.
- **Migration work** — Technical work involved in moving systems, services, or data from one setup to another. It often signals project ownership, coordination, and operational complexity.
- **MLOps engineer** — An engineer focused on the systems and processes that help machine learning models get deployed, monitored, and updated reliably. The role often overlaps with platform and infrastructure engineering.

- **Models** — In AI and machine learning, algorithms trained on data to make predictions or generate outputs. Recruiters should clarify whether a role builds models, adapts them, or uses existing ones.
- **Platform Engineer / Platform engineering** — An engineering role or discipline focused on building internal tools and infrastructure that other engineers use. It often sits near DevOps or SRE but is not always the same thing.
- **Polished transcript** — A cleaned-up written version of an interview or conversation. In this book, the term is used to warn against notes that sound tidy but lack judgment.
- **Product features** — The visible capabilities or functions users interact with in a software product. Some AI-related roles focus on shipping features that call external or internal APIs.
- **Production** — The live environment where real users interact with software. Experience in production usually signals operational responsibility and real-world impact.
- **Python** — A programming language widely used in backend development, scripting, data work, and machine learning. It is often referenced as a required or preferred skill in technical roles.
- **Queries** — Requests made to a database to retrieve or manipulate data. Speeding up queries is one reason database indexing matters.
- **Reliability** — The ability of a system to stay stable, available, and perform consistently in production. It is a key focus in SRE, platform, and infrastructure-related roles.

- **Recruiter screen** — An early-stage conversation between a recruiter and a candidate used to assess fit, motivation, and role alignment. In technical hiring, accuracy in this step matters more than sounding highly technical.
- **Rollout** — The process of releasing a change, feature, or system to users or production. Candidates who anticipate rollout risks often show stronger seniority.
- **Rollout failures** — Problems that happen during or after releasing software changes. Handling these well can indicate production experience and operational ownership.
- **Résumé** — A candidate's written summary of experience and skills. In this text, it is mentioned in the context of AI-driven filtering and candidate trust.
- **Scaled** — Improved a system so it could handle more usage, traffic, or complexity. This verb often signals meaningful technical impact.
- **Scalable** — Able to handle growth in users, data, or workload without breaking down. It is commonly used in architecture discussions, though often too vaguely.
- **Scorecards** — Structured evaluation forms used by interviewers to assess candidates against the same criteria. They help make technical hiring feedback more consistent and comparable.
- **Senior backend engineer** — A more experienced backend engineer expected to make stronger technical decisions and handle greater ownership. "Senior" should be tied to scope and decision-making, not just years of experience.

- **Site Reliability Engineer (SRE)** — An engineer focused on keeping systems reliable, stable, and available in production. The role often includes incident response, performance, and reliability engineering.
- **Software engineer** — An engineer who designs, builds, tests, and maintains software systems. It is a broad role that can include backend, frontend, platform, or other specialties.
- **Stack** — The combination of technologies, tools, and languages a team uses to build and run software. Recruiters should understand how the stack relates to the actual work, not just repeat the list.
- **System design** — The process of planning how a software system should be structured and how its components interact. It is also a common topic in technical interviews.
- **Service boundaries** — The lines that define what each service in a larger system is responsible for. Designing these boundaries usually signals architecture ownership.
- **Services** — Independent software components that perform specific functions within a larger system. In modern architectures, teams may own one or many services.
- **Stakeholders** — People with an interest in the outcome of a project or hiring process. In technical work, this often includes engineering, product, operations, customers, or leadership.
- **Terraform** — A tool used to define and manage infrastructure through code. It is often seen in cloud, DevOps, and platform engineering roles.

- **Tools** — Software used to support technical work, such as building, testing, deploying, monitoring, or analyzing systems. In recruiting, tools listed in a job should be tied to actual responsibilities.
- **Train / Training models** — Creating a machine learning model from data so it can perform a task. This is different from using or fine-tuning an existing model.
- **Transcripts** — Written records of spoken interviews or conversations. AI tools may generate transcripts, but recruiters still need to interpret them accurately.
- **Zoom** — A video meeting platform commonly used for recruiter screens, intake meetings, and interviews. In this text, it appears as part of the hiring workflow rather than as a deep technical term.

About This Book

About the Author

Natalia Romanova

Natalia is the founder of Talantrix, an applicant tracking system for tech recruiters. She began her career as a software engineer, later moved into engineering management, and eventually led several software companies, including ones she founded. Along the way she repeatedly had to build technical recruiting capability, and over time ended up learning a lot about hiring engineers—more than she ever expected to.

About Talantrix

Talantrix is an intelligent applicant tracking system built by recruiters, for recruiters. Designed specifically for technical hiring, it helps recruitment teams automate the busywork — from CV parsing and skill-based search to pipeline management — so they can focus on what matters most: connecting with top engineers and making great hires.

Learn more at https://talantrix.com (https://talantrix.com)

The Tech Recruitment 101 Series

A practical book series for recruiters who work with engineering teams. Each volume tackles a specific challenge in technical hiring — from screening candidates to understanding code — with clear explanations, real-world examples, and actionable frameworks.

Revision 1.0 — 19 May 2026

Tech Recruitment 101

A practical book series for recruiters who work with engineering teams. Each volume tackles a specific challenge in technical hiring — from screening candidates to understanding code — with clear explanations, real-world examples, and actionable frameworks.