Dari Junior ke Principal

Roadmap 15 Tahun dalam 15 Halaman

Bukan teori. Bukan kurikulum. Ini peta perjalanan nyata saya dari seseorang yang sudah melewatinya dan kamu bisa shortcut-nya.

Achmad Fajar Ridwan

Principal Engineer · AWS Certified Solutions Architect Professional
Pernah memimpin 56 engineers lintas negara · Ship Agentic AI MVP dalam 5 bulan Founder, Zero to Prod

https://zerotoprod.id · Maret 2026

Surat dari Fajar

Tahun 2011. Baru lulus kuliah, langsung kerja jadi PHP developer di Bandung.

Waktu itu saya pikir kalau saya bisa coding, bisa deliver task, ya sudah, itu cukup. Tapi makin lama kerja, makin ada perasaan yang mengganjal. Saya lihat senior engineer di sekitar saya dan sadar, ada sesuatu yang mereka punya yang saya tidak punya. Bukan soal hafalan syntax. Bukan soal tahu lebih banyak framework.

Mereka **berpikir berbeda.**

Cara mereka nanya pertanyaan. Cara mereka debug. Cara mereka push back ke product. Cara mereka bicara soal sistem, bukan hanya soal fitur. Itu yang beda.

Saya butuh waktu bertahun-tahun untuk mulai mengerti apa yang sebenarnya membedakan engineer yang cepat naik level dengan yang stuck di tempat yang sama.

PDF ini adalah peta perjalanan yang saya ingin miliki 14 tahun lalu.

Bukan kurikulum. Bukan teori dari buku teks. Ini adalah hal-hal yang saya pelajari sambil jalan, dari PHP developer, ke Java developer, ke DevOps engineer, ke team lead, sampai akhirnya jadi Principal Engineer yang memimpin 56 engineers lintas Indonesia, Australia, dan Filipina.

Setiap insight di sini punya harganya. Ada yang dari keputusan yang salah. Ada yang dari production incident jam 2 pagi. Ada yang dari percakapan jujur dengan engineer senior yang mau repot-repot kasih feedback keras.

Fajar

Principal Engineer & Founder, Zero to Prod

Baca ini dengan mindset terbuka. Ambil yang relevan, buang yang tidak. Dan kalau ada satu hal yang mengubah cara kamu melihat karir kamu — saya sudah senang.

The Big Picture: 5 Level Engineer

Sebelum kita bicara roadmap, kita perlu sepakat tentang peta-nya. Kebanyakan developer hanya tahu dua level: Junior dan Senior. Padahal dunia engineering jauh lebih bernuansa dari itu.

Level	Definisi Singkat	Fokus Utama
Junior Engineer	Bisa execute task yang sudah didefinisikan	Belajar, deliver, jangan break production
Mid Engineer	Bisa solve problem tanpa harus selalu ditunjukkan	Ownership, depth, mulai punya opini teknis
Senior Engineer	Bisa lead solusi teknis dan mentoring orang lain	System thinking, trade-off, multiplier
Staff / Principal	Bisa shape direction teknis tim dan produk	Strategy, architecture, alignment
Distinguished / Fellow	Pengaruh di level organisasi atau industri	Vision, thought leadership, legacy

> *Level ini bukan soal title di LinkedIn. Saya kenal "Senior Engineer" yang masih berpikir seperti Junior dan "Junior" yang sudah berpikir seperti Senior. Level adalah cara berpikir, bukan badge.*

Level 1: Junior Engineer (Tahun 0–2)

Tugas utamamu bukan untuk brilliant. Tugas utamamu adalah untuk reliable.

Apa yang seharusnya kamu fokuskan:

- Deliver apa yang diminta, tepat waktu, tanpa harus diingatkan berkali-kali
- Belajar cara kerja codebase yang sudah ada, jangan langsung ingin rewrite
- Tanya pertanyaan bagus, bukan "ini gimana caranya" tapi "ini kenapa dibuat seperti ini"
- Debugging: belajar baca error message dengan teliti sebelum minta bantuan

Mistake terbesar Junior

Menghabiskan waktu belajar teknologi baru tanpa menguasai yang sudah ada di depan mata. *Depth beats breadth* di level ini.

Tanda kamu sudah siap naik level:

- Kamu bisa deliver task tanpa perlu constant supervision
- Kamu mulai bisa estimate waktu dengan akurat
- Senior engineer di timmu mulai minta opinimu, bukan hanya memberinya

Level 2: Mid Engineer (Tahun 2–5)

Ini adalah level paling berbahaya, karena kamu cukup tahu untuk merasa sudah tahu banyak.

Skill yang harus dikuasai

- Debugging production issues, bukan hanya di local, tapi di sistem yang live di production.
- Code review yang substantif, bukan hanya cari typo, tapi cari logical flaw
- Estimate yang jujur, termasuk mengakui ketika tidak tahu
- Mulai baca sistem secara keseluruhan, bukan hanya bagian yang kamu sentuh

The Mid-Level Trap

Ini adalah level di mana banyak developer stuck selama 5-7 tahun tanpa sadar. Kenapa? Karena kamu sudah cukup produktif untuk dianggap valuable, tapi belum secara aktif develop skill yang dibutuhkan untuk naik ke Senior.

Tanda kamu terjebak di Mid trap: setiap tahun kamu belajar tools baru, tapi cara berpikirmu tentang sistem tidak berubah.

Cara keluar dari trap: Stop hanya deliver features. Mulai deliver insight tentang sistem.

Level 3: Senior Engineer (Tahun 5–8)

> *Senior bukan soal umur atau tahun pengalaman. Senior adalah soal multiplier effect.*

Definisi saya tentang Senior Engineer: **seseorang yang membuat tim di sekelilingnya lebih baik.**

Tiga pergeseran mindset di level Senior:

1. **Dari "bagaimana cara melakukannya" ke "mengapa kita melakukannya"** Senior selalu menantang requirement, bukan hanya mengeksekusinya. "Apakah ini benar-benar yang dibutuhkan user?" adalah pertanyaan yang Senior boleh dan harus tanyakan.
2. **Dari "code yang benar" ke "sistem yang benar"** Code yang sempurna dalam sistem yang salah tetap adalah kegagalan. Senior berpikir tentang integration, failure modes, dan operability, bukan hanya correctness.
3. **Dari "saya bisa" ke "tim kita bisa"** Senior Engineer yang baik sering kali sengaja tidak mengerjakan sesuatu sendiri karena mendelegasikan dan mentoring lebih valuable daripada mengerjakan sendiri.

Production-first thinking:

Kalau ini fail di production jam 3 pagi, siapa yang akan kena? ▸ Apa rollback plan-nya? ▸ Apakah kita bisa detect kalau ini bermasalah sebelum user merasakan?

Level 4: Staff / Principal Engineer (Tahun 8–12+)

Di level ini, kamu tidak lagi dinilai dari kode yang kamu tulis. Kamu dinilai dari keputusan yang kamu buat.

Ini adalah level yang paling tidak ada panduannya di internet karena sangat sedikit orang yang mencapainya, dan mereka yang mencapainya jarang menulis tentangnya.

Senior Engineer	Staff / Principal Engineer
Bertanggung jawab atas solusi teknis	Bertanggung jawab atas direction teknis
Bekerja dalam tim	Bekerja across tim
Delivery-focused	Strategy + delivery
Diukur dari output	Diukur dari outcomes
Menyelesaikan masalah	Memastikan masalah yang benar yang diselesaikan

Skill baru yang harus dikembangkan:

- **Technical governance** membuat keputusan arsitektur yang bertahan lama
- **Stakeholder alignment** menjelaskan trade-off teknis kepada non-engineer
- **Team amplification at scale** bukan hanya mentoring 2–3 orang, tapi 20–50 orang
- **Strategic roadmapping** tahu kapan invest di tech debt, kapan prioritaskan velocity

The Skills Matrix

Hiring manager menilai kamu bukan dari apa yang kamu tahu, tapi dari **cara kamu berpikir**.

Skill Dimensi	Junior	Mid	Senior	Principal
Technical depth	Satu area	2–3 area	Cross-domain	Arch. vision
Problem scope	Task-level	Feature-level	System-level	Org-level
Communication	Lapor progress	Diskusi solusi	Influence peers	Align stakeholders
Ambiguity	Butuh spec jelas	Handle unknown	High ambiguity	Create clarity
Mentoring	Tidak diharapkan	Peer mentoring	Active mentoring	Coaching culture
Production	Dengan supervision	Self-supervised	Team accountability	Cross-team

Lihat kolom level saat ini kamu. Kemudian lihat kolom level berikutnya. Gap itulah yang harus kamu kerjakan, bukan belajar tools baru.

Framework Memilih Teknologi

> *Saya sering ditanya: "Harus belajar apa?" Pertanyaan yang lebih baik adalah: "Kenapa saya belajar ini?"*

Core Platform (Non-negotiable)

- Cara kerja jaringan (TCP/IP, HTTP, DNS)
- Cara kerja database (indexing, transaction, query plan)
- Cara kerja operating system (process, memory, I/O)
- Version control yang benar (Git bukan hanya commit dan push)

Domain Depth (Level-dependent)

- Backend: concurrency, caching, API design, database optimization
- Infrastructure: networking, security, availability patterns
- AI/ML Engineering: inference pipeline, RAG, agent architecture

Production Skills (Universally Required)

- Observability: logging, metrics, tracing
- Deployment: CI/CD, blue-green, canary, rollback
- Incident response: runbook, post-mortem, blameless culture

Sebelum belajar sesuatu yang baru, tanya:

Apakah ini akan solve masalah yang sudah ada di pekerjaanku sekarang? ? Apakah ini skill yang dipakai di production, atau hanya trending di Twitter? ? Apakah aku sudah menguasai fundamentalnya sebelum loncat ke tools-nya?

PRODUCTION WAR STORY #1

Incident yang Mengubah Cara Saya Berpikir

Production adalah guru terbaik tapi sayangnya bayarannya mahal.

Waktu itu kami mengelola infrastruktur healthcare SaaS yang melayani ribuan clinical users di Australia. Jam 2 pagi, alerting berbunyi. Platform down.

Root cause-nya sederhana: query yang tidak punya index yang tepat. Dengan load normal, query ini tidak masalah. Tapi ketika traffic spike mendadak, database kehabisan connection pool. Domino effect mulai, dalam 8 menit, platform tidak bisa diakses.

Apa yang saya pelajari:

1. Tidak ada yang namanya "ini tidak mungkin terjadi di production" Semua engineer yang merancang sistem itu kompeten. Sistem kompleks punya failure mode yang tidak selalu kelihatan di development.
2. Observability adalah asuransi, bukan luxury Kalau kami tidak punya monitoring yang benar, kami tidak akan tahu ada yang salah sampai user call support. Waktu 8 menit deteksi, bukan 80 menit, karena alerting yang tepat.
3. Runbook yang baik lebih berharga dari engineer yang panik Ketika alarm berbunyi jam 2 pagi, kamu tidak ingin berpikir dari nol. Kamu ingin ikuti checklist yang sudah disiapkan saat masih jernih.

Pertanyaan untuk setiap fitur yang kamu ship: Kalau ini fail, bagaimana caraku mendeteksinya? Bagaimana caraku me-rollback-nya?

PRODUCTION WAR STORY #2

Lesson dari Memimpin Tim Besar

Memimpin 56 engineers bukan soal menjadi yang paling pintar. Soal memastikan semua orang bisa bergerak ke arah yang sama.

Masalah teknis tersulit bisa diselesaikan dengan engineering yang baik. Masalah manusia butuh sesuatu yang berbeda

Saya pernah punya situasi di mana dua tim yang bekerja pada sistem yang saling terhubung memiliki understanding yang berbeda tentang kontrak API mereka. Keduanya deliver dengan baik di silo mereka. Tapi interface-nya tidak kompatibel dan baru ketahuan seminggu sebelum deadline.

Lesson:

1. **Alignment lebih awal, lebih murah** Satu jam architectural review di awal sprint bisa menyelamatkan dua minggu rework di akhir.
2. **Dokumentasi bukan overhead — itu adalah multiplier** Dengan 56 engineers lintas timezone, ADR (Architecture Decision Records) menjadi cara kami memastikan keputusan penting tidak hilang dalam email thread.
3. **Psychological safety bukan soft skill** Tim yang tidak berani bilang "gue tidak tahu" akan menyembunyikan masalah sampai terlalu besar, ini adalah isu teknis yang manifes sebagai kegagalan delivery.

> *Mulailah melatih satu skill ini sekarang, jauh sebelum ada title-nya: kemampuan untuk membuat orang lain lebih efektif, bukan hanya dirimu sendiri.*

The AI Engineering Shift

AI bukan trend yang bisa kamu abaikan. Tapi juga bukan alasan untuk melupakan fundamentals.

Di 2025, saya single-handedly ship sebuah Agentic AI platform dari nol ke production dalam 5 bulan: LangGraph, RAG pipeline, multi-agent orchestration, Temporal workflow, full-stack TypeScript + Python, deployed di AWS ECS.

1. **AI engineering tetap butuh software engineering yang baik** Banyak developer loncat ke AI tanpa fondasi solid. Hasilnya: AI demo yang impressive tapi tidak bisa di-maintain, tidak bisa di-scale, dan tidak bisa di-debug saat production error.
2. **Non-determinism adalah tantangan fundamental yang berbeda** Software tradisional: input sama → output sama. AI: input sama → output BERBEDA setiap kali. Ini mengubah cara kamu test, monitor, dan define "success."
3. **Prompt engineering adalah skill yang nyata** Cara kamu struktur prompt, cara kamu define agent tools, cara kamu handle multi-turn context, semua adalah engineering decisions dengan tradeoff nyata.

Rekomendasi masuk AI space:

Tahap	Fokus
1. Kuasai dulu	REST API, async programming, database basics
2. Kemudian pelajari	LLM fundamentals, embedding, vector database
3. Baru eksperimen	Agent framework: LangGraph, CrewAI, dll

Roadmap Konkret: Dari Level ke Level

Berhenti tanya "apa yang harus saya pelajari." Mulai tanya "level mana yang saya target dalam 12 bulan ke depan."

Junior → Mid Target: 18–24 bulan dari mulai kerja

- Deliver 3 fitur end-to-end tanpa supervision
- Debug dan fix setidaknya satu production issue sendiri
- Lakukan code review substantif untuk minimal 10 PR
- Baca dan pahami arsitektur sistem yang kamu kerjakan (bukan hanya bagianmu)

Mid → Senior Target: 2–4 tahun sebagai Mid

- Lead delivery satu proyek/feature dari awal sampai production
- Mentoring minimal satu Junior Engineer secara aktif
- Buat keputusan arsitektur yang kamu bisa defend di hadapan tim
- Tulis runbook untuk sistem kritis yang kamu maintain

Senior → Principal Target: 3–5 tahun sebagai Senior

- Shape technical direction untuk satu product area
- Align minimal 2 tim berbeda pada keputusan teknis lintas-tim
- Jadi "go-to person" untuk satu domain teknis di organisasi
- Mulai bicara dalam bahasa bisnis, bukan hanya bahasa teknis

Tentang Karir di 2026 dan Seterusnya

Dunia berubah. Yang tidak berubah adalah nilai dari engineer yang bisa berpikir.

1. **AI tidak menggantikan engineer, tapi engineer yang pakai AI akan menggantikan yang tidak** Engineer yang bisa gunakan AI tools secara efektif menghasilkan output 2–3x lebih cepat. Gap ini akan terus membesar.
2. **Remote work membuka global market, tapi juga memperketat kompetisi** Sebagai developer Indonesia, kamu sekarang bisa bersaing untuk posisi yang membayar gaji global. Tapi artinya kamu juga bersaing dengan developer global yang punya akses ke resource yang sama.
3. **Spesialisasi makin penting** Era "full-stack yang bisa semua" semakin tergantikan oleh "deep expertise di satu area, cukup tahu di area lain." Tentukan T-shape-mu: apa yang kamu kuasai sangat dalam?

Keunggulan yang harus kamu bangun:

- Domain expertise yang dalam (bukan generalis yang tahu sedikit banyak hal)
- Communication yang baik dalam Bahasa Inggris, ini adalah leverage yang besar
- Portfolio kerja nyata, bukan hanya sertifikasi
- Presence online, LinkedIn, GitHub, atau content yang menunjukkan cara berpikirmu

12 bulan dari sekarang, apa satu kemampuan konkret yang ingin orang lain asosiasikan dengan namamu?

Penutup: What's Next

Ini hanyalah awal dari percakapan.

Kalau kamu sampai di halaman ini, terima kasih sudah membaca. Saya harap setidaknya satu hal dari PDF ini memberimu kejelasan tentang di mana kamu sekarang, ke mana kamu ingin pergi, dan langkah konkret pertama yang bisa kamu ambil.

Apa yang akan saya bahas lebih dalam di Zero to Prod:

- Cloud Architecture: bagaimana kamu membuat keputusan infrastruktur di production nyata
- System Design: framework yang benar-benar dipakai, bukan hanya untuk whiteboard interview
- DevOps dari perspektif engineer: CI/CD, observability, dan incident response
- AI Engineering untuk production: LangGraph, RAG, agent architecture dari nol sampai ship

Satu langkah yang bisa kamu ambil sekarang:

Gabung ke WhatsApp Group Zero to Prod. Di sana saya share insight, diskusi, dan update langsung dari lapangan — bareng developer Indonesia yang juga lagi dalam perjalanan yang sama.

zerotoprod.id/whatsapp

Sampai ketemu di sana.

— Fajar *Principal Engineer · Founder, Zero to Prod*