Why Basic Knowledge is the Key: A DevOps Engineer's Take on the AI Era

11 min read · 4 days ago

Quan Huynh Follow

Understanding fundamentals matters more than ever when AI can generate everything!

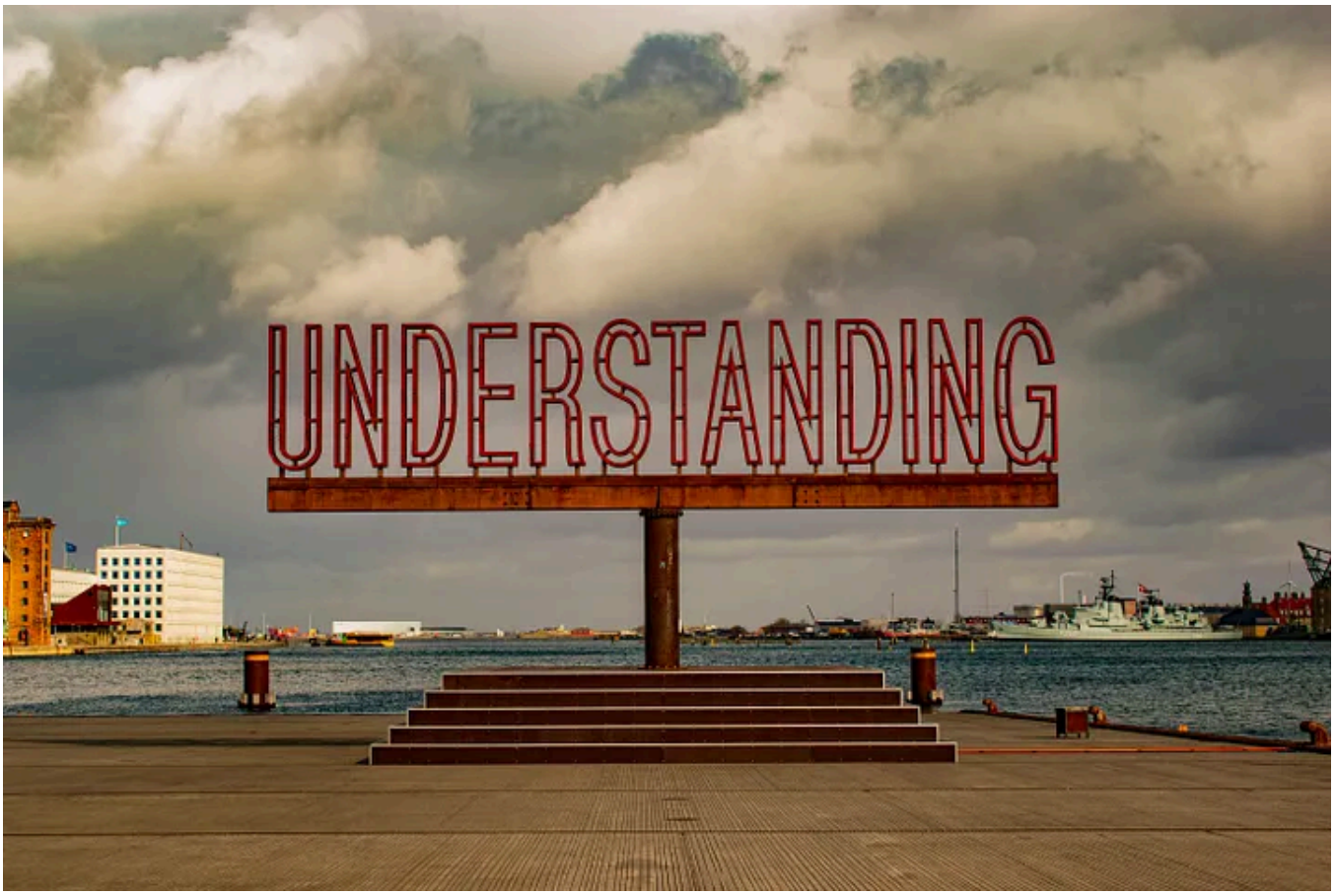

Image of Zuzana Ruttkay

The software industry is in the middle of a profound shift, and most of us are still catching up to what it means.

AI tools can now generate infrastructure code in seconds. GitHub Copilot, ChatGPT, and Claude can scaffold a complete Terraform module, write Kubernetes manifests, or build CI/CD pipelines faster than you can Google the syntax. On the surface, this looks like pure productivity gain. We're shipping faster. Iterating more. Moving at a pace that would have seemed impossible just two years ago.

But speed without direction is just motion.

Here's what I've learned after watching teams — including my own — navigate this new world: **AI doesn't replace the need for deep understanding. It amplifies the consequences of not having it.**

The Shift We're Not Talking About

Last week, I asked Claude Sonnet to generate a Terraform module for a "production-ready" Kubernetes cluster. Three minutes later, it was written. Five minutes after that, it deployed successfully. The velocity felt incredible. Everything worked perfectly.

But here's what I didn't see — because I didn't know to look for it:

No resource limits on containers. Security groups are wide open to `0.0.0.0/0`. No backup strategy. No monitoring. No disaster recovery plan. The cluster would accept any workload size until memory ran out. An attacker could SSH from anywhere. We had no way to recover from data loss.

The code worked. It deployed. But it was a ticking time bomb.

> *And here's the frightening part: without fundamental knowledge, I couldn't tell the difference between "it works" and "it's production-ready."*

The AI gave me exactly what I asked for. But I didn't know the right questions to ask. I didn't understand what production-ready actually meant. I couldn't spot the dangerous gaps because I was unaware of their existence.

We've entered an era where tactical work — such as writing configuration files, deploying resources, and setting up services — can be automated. What can't be automated is the judgment about whether we should do those things, how they fit into the bigger picture, and what happens when they fail.

And if you're a DevOps engineer, this shift matters more for you than almost anyone else.

Because when application code fails, users see an error. When infrastructure fails, everything stops.

What We're Missing

> *The Fundamentals*

I've been watching this pattern repeat across the industry. Engineers are deploying infrastructure without understanding how networks actually route traffic. Teams running Kubernetes in production who can't explain what a container really is. Security configurations copied wholesale from AI suggestions, with nobody questioning whether they're appropriate for the use case. Monitoring gets added last, if at all, because the prompt focused on deployment and forgot observability entirely.

The core issue isn't the tools or the AI. It's that we've created an environment where you can ship infrastructure to production without ever learning why it works. And that's terrifying.

AI amplifies what you know. If you don't know the basics, AI will just help you fail faster.

Picture this: it's 3 AM, and that AI-generated Kubernetes cluster just went down. Your pager is screaming. Your team is panicking. And suddenly you need to understand how container networking actually functions, what resource limits mean and why they matter, how load balancing and service discovery interact, and what happens when etcd runs out of disk space.

AI can't debug production for you at 3 AM. Your fundamental knowledge can. Or more accurately, your lack of fundamental knowledge will be what keeps you and your team up all night.

Why Basic Knowledge is Your Key

You Can Ask the Right Questions

AI is only as good as your prompts. And your prompts are only as good as your understanding of what you actually need.

When someone without fundamental knowledge asks AI to “generate a Kubernetes deployment,” they get something that technically works. When someone with a deep understanding asks AI to “generate a Kubernetes deployment with resource limits based on 2GB memory and 1 CPU, include liveness and readiness probes checking /health every 10 seconds, set replicas to 3 for high availability, and use rolling update strategy with maxUnavailable set to 1,” they get something that actually survives production.

The difference isn’t the AI. It’s knowing what production deployments need in the first place. It’s understanding that high availability requires multiple replicas, that health checks prevent routing to broken pods, that resource limits stop one service from starving others, and that rolling updates keep your service available during deployments.

You Can Spot Dangerous Mistakes

AI doesn’t understand your context. It doesn’t know your business constraints, your scale, your security requirements, or your operational capabilities. It just generates code that matches the pattern it’s seen before.

I’ve caught AI suggesting t2.micro instances for a database that handles 10,000 transactions per second. I’ve seen it open SSH to the internet “for convenience.” I’ve reviewed auto-scaling configurations that would have bankrupted companies by spinning up hundreds of expensive instances. I’ve found backup strategies that looked good on paper but couldn’t actually restore data because they didn’t account for our specific data model.

How did I catch these? Not because I’m smarter than AI. But because I understand compute capacity planning well enough to know a t2.micro can’t handle that load. Because I’ve learned security principles and know that convenience and security are often at odds. Because I’ve seen what happens when auto-scaling goes wrong and learned to check the math on worst-case costs. Because I’ve actually restored from backups and know the difference between “we’re backing up” and “we can recover.”

You Can Make Tradeoffs

AI gives you “a solution.” But every infrastructure decision involves tradeoffs. Performance versus cost. Availability versus complexity. Security versus convenience. Speed versus reliability.

AI can't make these decisions because it doesn't know your constraints. It doesn't know that your budget is tight this quarter. It doesn't know that your team is already stretched thin and can't handle operational complexity. It doesn't know about your compliance requirements or industry regulations. It doesn't know that your business priority this month is stability over features.

These tradeoffs require judgment. And judgment requires understanding the implications of each choice.

You Can Prevent Problems Before They Happen

The most valuable engineers aren't the ones who fix problems quickly. They're the ones who prevent entire classes of problems through thoughtful design.

This requires understanding things that AI has never experienced: the common ways distributed systems fail, the attack vectors that security teams worry about, the performance bottlenecks that appear at scale, and the operational complexity that makes systems unmaintainable.

AI can help you implement solutions. But fundamental knowledge helps you design systems that don't need those solutions in the first place.

The Fundamentals That Matter Most

Let me be specific about what "fundamentals" actually means, because it's not about memorizing commands or collecting certifications.

Networking is where so many infrastructure problems hide. You need to understand how TCP/IP actually works, not just know that it exists. You need to grasp what DNS is doing behind the scenes when services can't find each other. You need to know the different load balancing strategies and when each makes sense. Network security and segmentation need to be more than buzzwords — you need to understand why these boundaries exist and what they're protecting against. AI can generate network configurations all day long, but when services can't reach each other, you need to understand why.

Linux and operating system fundamentals become critical the moment things go wrong. How processes, memory, and CPU interact isn't academic knowledge — it's what you need when your container gets OOM killed at 2 AM. File systems and disk I/O matter when you're trying to figure out why your database is slow. User permissions and security aren't just checkboxes — they're how you prevent one

compromised service from taking down your entire infrastructure. Understanding system signals and process management is what separates engineers who can debug production from those who just restart things and hope.

Distributed systems concepts are no longer optional in modern infrastructure. The CAP theorem isn't trivia — it's the explanation for why your database behaves the way it does under network partitions. Failure modes and fault tolerance design determine whether your system recovers gracefully or cascades into total failure. The difference between stateful and stateless design affects everything from your deployment strategy to your scaling approach. Data replication and synchronization determine whether you can actually recover from disasters. AI doesn't understand that eventual consistency might break your business logic — you need to.

Security principles can't be prompted into existence. Defense in depth isn't just "use multiple tools" — it's a philosophy of layered protection where each layer assumes the others might fail. Least privilege access isn't bureaucracy — it's limiting blast radius when (not if) something goes wrong. Authentication versus authorization might seem like semantic difference until you realize you've been doing one when you needed both. Encryption at rest and in transit aren't optional extras — they're baseline protection against classes of attacks. Security isn't something you can prompt your way out of.

Cost and capacity planning separate engineers who ship from engineers who ship sustainably. You need to understand how cloud pricing actually works, not just the sticker prices but the hidden costs of data transfer, API calls, and state storage. Resource utilization and waste matter more as scale increases — a 10% improvement in utilization might be millions of dollars. Scaling economics aren't linear, and understanding where the cost curves change helps you make better decisions. TCO thinking means considering not just deployment cost but operational cost, maintenance cost, and the cost of eventual migration. That AI-generated multi-region architecture might cost $50,000 per month — do you even know how to calculate that?

How to Build Your Foundation

The path to fundamental knowledge isn't mysterious, but it does require intention. Let me share what I've learned actually works.

Start with the basics, not the shiny tools. Everyone wants to jump straight to Kubernetes. I get it — it's what the job postings ask for, it's what looks impressive on

your resume. But here's what I tell every engineer I mentor: before you touch Kubernetes, learn to deploy an application manually on a Linux server. Figure out how to make it resilient — what actually happens when it crashes? Build monitoring so you know it's working. Then scale it manually and discover what breaks first when traffic increases. Now — only now — use Kubernetes. Because now you understand what problems it's actually solving. You'll know why it makes those tradeoffs. You'll recognize when it's overkill for your use case.

Learn by breaking things. This is non-negotiable. Set up a lab environment and deliberately destroy it. Delete critical files and then recover. Simulate network failures. Fill up disk space. Exceed memory limits. Corrupt databases. I know it sounds wasteful, but you learn more from one hour of fixing a broken system than from ten hours of reading about systems that work. The panic you feel when everything's on fire and you have to figure out why — that's where real learning happens. Better to experience that panic in your lab than in production at 3 AM.

Understand the "why," not just the "how." When you're learning, constantly ask yourself why things work the way they do. Not "how do I deploy this?" but "why does this deployment strategy prevent downtime?" Not "how do I add health checks?" but "why do we need health checks in the first place?" Not "how do I set resource limits?" but "why are resource limits important, and what happens without them?" AI can tell you how to do things. But experience — and curiosity about deeper principles — teaches you why. And why it matters more.

Read the actual documentation. I don't mean blog posts. I don't mean AI summaries. I mean the official Kubernetes documentation that explains design decisions. AWS architecture whitepapers that discuss tradeoffs. RFCs for protocols you use that explain why they work that way. Source code for tools you depend on that reveals what they're actually doing. Documentation explains the reasoning behind decisions. It discusses the tradeoffs the designers considered. It reveals the edge cases and limitations. AI summaries give you enough to use a tool. Documentation gives you enough to understand it.

The Real Future of DevOps

Software development is shifting from writing code to designing systems. Developers are becoming architects — not in the enterprise title inflation sense, but literally: their job is to design the structures within which code gets created.

In DevOps, we're experiencing the exact same evolution.

But here's what makes this different for us:

When a React component fails, users see an error page. When infrastructure fails, everything goes dark. The stakes are higher. The margin for error is smaller. The consequences of getting it wrong are more severe.

You can't architect resilient infrastructure without understanding how systems fail.

You can't design secure networks without understanding attack vectors.

You can't make cost-effective decisions without understanding cloud pricing models.

You can't prevent incidents without understanding failure modes.

Basic knowledge isn't just helpful. It's the foundation everything else is built on.

The engineers who thrive in the AI era won't be the ones who can prompt the fastest or generate the most code. They'll be the ones who understand systems deeply enough to:

- Ask the right questions
- Spot dangerous mistakes before deployment
- Make informed architectural tradeoffs
- Design systems that are resilient by default
- Encode their knowledge into frameworks that guide AI safely

What This Means for Your Career

The work is changing. Not disappearing — changing in fundamental ways that will separate those who thrive from those who struggle.

If your value proposition is knowing tool syntax, AI is already better at that than you'll ever be. If it's following tutorials, AI can do that instantly without getting distracted or tired. If it's copying and pasting configurations, that work is already automated and getting more automated every day.

But if your value is understanding systems deeply — really understanding them, down to how they fail and how they scale and what tradeoffs they involve — you're more valuable than ever. If your value is making informed tradeoffs between

competing constraints, you're irreplaceable because AI can't know your constraints. If your value is preventing problems through design rather than heroically fixing them, you're essential because prevention requires judgment that AI doesn't have. If your value is encoding your operational wisdom into platforms and frameworks that guide others, you're building the future.

The bottleneck is no longer speed. Its direction.

We can generate infrastructure faster than ever before. We can deploy changes in minutes that used to take days. We can spin up complexity instantly. But speed without direction is just motion. The real questions are: Are we building the right things? Do they work together coherently? Will they scale gracefully or catastrophically? Are they secure against real threats? Can we actually afford them at scale? Can we operate them effectively with our current team?

These questions require judgment. And judgment requires understanding. Not surface-level familiarity, but deep comprehension of how systems actually work, why they fail, and what it takes to run them sustainably.

> *Basic knowledge isn't boring. It's your competitive advantage. Start building that foundation today.*

. . .

If you want to learn more about building AI infrastructure agents. How do you implement MCP servers? How do you manage infrastructure state for AI agents? You can find it in "The AIOps Book".

DevOps AI Sre Cloud Computing

Follow

Written by Quan Huynh

1.9K followers · 45 following

The AIOps Book: https://leanpub.com/the-aiops-book

Responses (1)

Bgerby

What are your thoughts?

See all responses