

ESSAY / 2026.10

The Distracted Forward Deployed Officer

BAZYL HORSEY CHICAGO, IL

I had more hair

When I met Arvist, I had more hair. I don't remember much of the four years that followed. I was distracted.

I joined as Arvist's first technical hire and became its cofounder and CTO. I was a hired gun. Young, a little dangerous, looking for something to point at and shoot. My last couple of startups hadn't made it. Arvist was the first one I couldn't let die.

If you haven't heard of Arvist: it turns the cameras a warehouse already has into computer vision that watches the floor. Inspections, safety, the moments nobody can watch all day. It installs in a day and works with the systems a site already runs.

I've never been the type A guy, and I won't pretend to be one now. I'm type "find a way." I wrestled in middle school, high school, and part of college. Most of what I know about running engineering came from the mat: stay low, keep moving, fight for position. But Nilay, my cofounder and Arvist's CEO, noticed two things about me. I had my head in the clouds. And I would find a path around any obstacle, pushing back only if a request wasn't profitable for the one customer asking *and* had no chance of ever serving anyone else.

That second habit turned out to be the job. It had a name somewhere. I just hadn't read it yet.

Every warehouse is different

I came in talking about multiples. Build it once, sell it many times. That's the whole promise of SaaS.

Nilay answered in four words: "Every warehouse is different."

I listened. Then I had an existential crisis. How do you build a SaaS when every customer is different?

There was no time to find the perfect answer. We had customers to deploy. So we deployed, and let the answer find us.

A hundred pounds of sand

One government-adjacent project needed us to detect sensitive powders on a facility floor. Explosive ones. The kind you don't spill for practice. I had no business being anywhere near them.

So I bought 100 pounds of colored sand and poured it across the floor. My team watched the cameras, recording the RTSP streams through VLC. I swept the sand around. All day.

You could read the question on their faces: what the hell is he doing? To their credit, they stayed along for the ride. I didn't fully know what I was doing either. I was distracted.

By the end of the day, we had training data. That's forward deployment in miniature: the customer's problem, a bag of sand, and whatever gets you to a working model.

I wasn't the guy with the Gantt chart. I was the guy with the sand.

Another time, we needed a safety incident on camera. A real-looking one. So I played the distracted, unsafe worker. Headphones on. Arms full of stacked boxes I couldn't see over. I walked straight into another pile of boxes and went down. On purpose. I landed on a box corner and bruised a rib.

That was kind of the point. It had to look real. I knew I could take the punch, I knew everything would be okay, and I knew Arvist would have good evaluation data by the end of it. For once, I was only pretending to be distracted.

It wasn't the only room I had no business being in. I ended up defining KPIs for DoD-funded work alongside professors who knew far more than I did. A very thorough government PM once caught me holding a screwdriver that wasn't on our funding sheet. I'd borrowed it from the next desk. I was distracted.

The part that isn't different

Multiples did show up. Just not where I expected. Not in features. In protocols.

Yes, every warehouse is different. But nearly every camera speaks RTSP. Nearly every system worth talking to speaks HTTP or MQTT. So I drew a hard line: we build on open, agnostic protocols, and we don't spend our weeks on proprietary APIs.

That hit some nerves. It was still right. When you can't standardize the customer, standardize the plumbing. Custom work gets cheap when the boring layer never moves.

A reason to never say no

Time moved. I was distracted. Customers came, and it was a big, beautiful blur of getting shit done.

Somewhere in that blur, I gave our dev team a speech I still believe:

A smart dev team knows when to push back. A genius dev team gives the business a reason to never say no to the customer.

That isn't the same as having no backbone. Saying yes to everything is how teams burn out. Building systems where yes is cheap is how teams win. Serve the customer by being smart, not by being a pushover.

Pushback is a privilege you earn by shipping.

Then a customer said something I'll keep forever: "Wow, these guys can do everything." They were talking about my dev team.

What I was actually listening to

Here's what I heard, all those years I looked like I wasn't paying attention.

Nilay told me every warehouse is different. I heard a product spec, not a warning.

Customers told me their compliance requirements were unavoidable. A sensitive site doesn't bend its procurement for a startup. So we became the kind of company that gets through it.

Customers told me they needed KPIs. Not a feed full of detections. Numbers they could carry into their own meetings and defend.

And customers told me, without ever saying it out loud, that uptime was ours. Over 90% of our incidents traced back to site power, network, or hardware, not our code. It didn't matter. You can't ask a customer to share responsibility for their own building. You can't write a runbook for one, either. So we owned all of it.

Even the acts of God. When a hurricane came through, I felt like Odysseus: the gods throwing weather at us, and me just trying to get everything back online and get home.

None of that came from a playbook. It came from listening.

Carrying it

At first, the job was comfortable. I was the first committer. I drew the architecture and taught our first team the basics: git team workflows, how to work in one repo together, how to use Docker. I was nurturing a team, and the problems still fit in my head.

Then there was the other job. It felt like constant failure, just pointed forward. Reality punched me in the face, personally and often. Half the time I thought I was drowning. Turns out I was swimming. It looks exactly the same from inside the water.

Every customer was different. The product lived in someone else's building, on someone else's power and network. That's about the hardest kind of company a small pre-seed startup can be. It's also the moat. Something this hard to make is just as hard to copy, and we were making it, successfully. My developers didn't need that distraction. So I carried it.

I didn't protect them from the work. I protected them from the weight. Some weeks the only thing holding it up was my belief in my own strength and tenacity.

You don't get to be a cofounder and also be tired of it. Some mornings I had to go find something in myself to keep it breathing, because nobody else was coming to do that part. And I'll say it plainly: I loved it. I was passionate the whole way through, even on the drowning days.

I didn't know what I was doing. I'm saying that on purpose. The people who say they did either weren't in the room or are lying.

Then came compliance. Customers required it, Nilay needed it, and they were right: high-sensitivity customers don't get through procurement on charm. So I took it on, and became the security lead of a company that had never had one.

What I didn't have was the bravery to tell him how much stronger I'd need to become to carry it. Compliance, on top of nurturing a team, on top of writing code and reviewing it, on top of strategy and product. Any one of those is a job. I was doing all of them and trying not to let anyone see the seams.

That was the era when I couldn't afford to be distracted. In hindsight, it's when I was most distant.

That's the difference I didn't understand at the time. Distracted, my mind went to the floor, the customer, the team. Distant, it went nowhere. It just held on.

I'll own the misses from that era. I won't apologize for the swings.

This is where I was.

Where the limit was

When we got in over our heads, it was because we'd hit a real limit, and the answer was structure. We turned what our customers asked for on ourselves: KPIs in live feedback loops, in real time, in a team small enough that everyone could see them.

What it turned into

The big players who coined "forward deployed engineer" mostly mean an engineer embedded with a customer, writing code on site. That's part of it. We weren't one of those players, and what we built at Arvist got there by a different road.

Palantir had already written this playbook. We didn't have a copy. So we wrote it again, from scratch, on warehouse floors across the country. I was on planes within weeks of starting.

When every AI startup started hiring "forward deployed engineers," I read the job descriptions and recognized our Tuesdays.

We didn't copy the playbook. We were too busy to read it. Same pressure, same answer. That isn't luck. That's the answer being right.

For us, forward deployment became a unit of development. Customer-need led. People in the loop. And an amazing product manager who could tell when a feature we'd built three times needed to graduate into a stable, traditional product.

Today Arvist has landed a Fortune 500 customer. It runs a partner program that reaches across the world. Those partners can forward deploy without us in the loop, because the toolbelt lets them:

- **Frontend SDKs**, so we can build into the dashboard screens customers already have on their floors.
- **Backend SDKs**, so anyone can build on our streaming pipelines without re-solving primitives like motion detection or OCR.

I don't need to be in the room for it to keep working anymore. That was the whole point of the toolbelt.

And the team still goes onto the floor. They get to know the operators and the executives. They learn the pains, and they deliver.

We got dirty. I have scars. It was worth it.

The playbook, as I'd hand it over

If I could give the next first technical hire one page, it would be this.

01 Hear what's under the ask. Every warehouse is different. Compliance, KPIs and uptime aren't.

02 Push back on one condition only. When it isn't profitable for this customer and has no chance of serving the next one.

03 Standardize the plumbing, not the customer. RTSP, HTTP, MQTT. Open protocols over proprietary APIs, every time.

04 Make yes cheap. A smart team knows when to push back. A genius team builds systems that give the business a reason to never say no.

-
- 05 **Productize on the third build.** When you've built it three times, hand it to someone who can make it stable.

 - 06 **Own the uptime, even when it isn't yours.** The customer can't share responsibility for their own building.

 - 07 **Turn the KPIs on yourself.** Live feedback loops, visible to the whole team.

 - 08 **Get on the floor.** Meet the operators and the executives. Learn the pains. Deliver.

 - 09 **Hand over the tools.** SDKs, so your own forward deployed engineers and your partners spend less time rebuilding the basics and more time making the customer happy.

Where my mind drifted

I'm not going to list the hundreds of things I owned at Arvist. That isn't the point, and it isn't the gift.

Here's the point. I looked distracted, and some days I was. But my mind wandered. It wandered to the floor. Every single time. To the operator fighting a screen that didn't help them. To the camera angle that would never see the pallet. To the customer who would say yes if we could just find a way to never say no.

Distracted is what listening looks like when the person you're listening to isn't in the room.

So here's the definition I'd offer, from a company that isn't one of the big players. Yet. A forward deployed engineer isn't defined by where they sit. They're defined by where their attention goes when nobody is directing it.

Mine went to the customer. It always did. I think the team I'm leaving behind does the same, and that's the thing I'm proudest of.

You raise something so it can stand without you. Nobody warns you how it feels when it works. It's a strange kind of proud.

Nilay told me every warehouse is different. He was right. So am I, now.

To the team: thanks for following the guy with the bag of sand.

This is the story of the distracted forward deployed officer. I was distracted. Turns out I was also listening.