Tech Jargon

Part 1

for

Product Managers

Where Your Product Lives

Cloud | Containers | Serverless | CDN

From an Engineer-Turned-PM

The Cloud & Platform Mental Model

A way to see **where your product actually runs** — cloud providers, regions, environments, containers — and how infrastructure choices shape cost, performance, and velocity.

It helps PMs have informed conversations about deployments and infrastructure. Get this wrong and infra conversations feel foreign. Get it right, and you understand why "just deploy it" is never that simple.

When you understand this mental model, you can:

- Understand the path from dev to staging to production — and why regions and CDNs shape user experience
- Participate in cloud cost conversations and know when to optimize vs. invest
- Recognize how containers and orchestration changed what "scaling" means
- Know why your team moved from VMs to containers — and what that unlocked

This guide covers the 10 essential terms that bring this mental model to life.

Cloud provider

The company that rents you servers

A cloud provider (AWS, Azure, GCP) rents you compute, storage, and networking on demand — so you don't buy and maintain your own hardware. You pay for what you use, scale when you need, and let someone else handle the hardware.

Renting an apartment vs. buying a house. You trade ownership for flexibility — no roof to fix, no furnace to replace, and you can move when your needs change.

We're on AWS, but the data team uses GCP for BigQuery — so we're multi-cloud whether we planned it or not.

For PMs, this means the cloud provider shapes what services are available, what regions you can deploy to, and what the bill looks like. You don't pick it — but you should know which one you're on.

Region / Availability zone

Where in the world your product runs

A region is a geographic location where a cloud provider has data centers — US East, EU West, Asia Pacific. Each region has multiple availability zones (AZs): physically separate buildings with independent power and networking, so if one goes down, the others keep running.

A restaurant chain with locations in different cities (regions). Each city has multiple kitchens (AZs) — if one kitchen has a fire, the others keep serving.

Latency is high for EU users because we're only deployed in us-east-1. We need to add a European region.

For PMs, this means region choices affect latency (how fast the product feels), data residency (where data legally lives), and cost. If users in a region complain about speed, the problem might be infrastructure, not code.

Environment

Dev, staging, production — the layers before users

An environment is a separate copy of your system used for a specific purpose. Dev is where engineers build and test. Staging mirrors production for final validation. Production is where real users live. Code moves through these layers before it reaches customers.

A dress rehearsal before opening night. Dev is practice in your living room. Staging is the full rehearsal with costumes and lighting. Production is the live audience.

It works in staging but breaks in production — the staging database doesn't have the same volume of data.

For PMs, this means it's deployed" doesn't mean users can see it. Ask which environment. And when staging doesn't match production closely enough, bugs slip through — that's not carelessness; it's an environment parity gap.

Container

A portable box that runs your code anywhere

A container packages your code, dependencies, and configuration into one unit that runs the same way everywhere — your laptop, staging, production. Unlike a VM (which emulates an entire operating system), a container shares the host OS and isolates the app layer. Lighter, faster, more portable.

A shipping container. It doesn't matter if it's on a truck, a train, or a cargo ship — the contents stay the same and the container fits on any platform built to carry it.

"We containerized the app so deploys are identical across environments — no more 'it works on my machine.

For PMs, this means containers are why modern teams can deploy fast and consistently. If your team moved from VMs to containers, they traded heavyweight machines for lightweight, portable units — and gained speed at every stage.

Orchestrator

The system that manages all your containers

An orchestrator (most commonly Kubernetes) automatically deploys, scales, and restarts your containers. If a container crashes, it spins up a replacement. If traffic spikes, it adds more. It turns "managing 200 containers by hand" into "declare what you want and let the system handle it."

An air traffic controller for containers. Planes (containers) take off, land, and sometimes have emergencies — the controller keeps them organized, routes them to open runways, and calls in replacements when needed.

Kubernetes auto-scaled the pods during the traffic spike — we didn't have to touch anything.

For PMs, this means when engineers say "Kubernetes handles that," they mean the system self-heals and scales automatically. It's powerful — but it's also complex. K8s migrations are significant investments, not quick wins.

CDN

Content delivery network — fast content worldwide

A CDN caches copies of your static content (images, CSS, JavaScript, videos) on servers distributed around the world. When a user in Tokyo requests your page, they get it from a nearby CDN server instead of your origin server in Virginia. Less distance, less latency.

A library system. Instead of everyone driving to one central library, every neighborhood gets a branch with copies of the popular books. Faster access, less traffic to the main building.

Static assets load in 50ms because they're on the CDN — but the API call still hits origin, so that's where the latency is.

For PMs, this means CDNs make your product feel fast for static content. If users complain about speed, ask whether the slow part is cached on the CDN or hitting the backend directly. The fix depends on which one is slow.

Serverless

Pay-per-execution compute — no servers to manage

Serverless (AWS Lambda, Google Cloud Functions) runs your code only when triggered — an API call, a file upload, a scheduled job. No servers to manage, no idle costs. But there are tradeoffs: cold starts add latency on first invocation, and costs can spike at high volume.

A taxi vs. owning a car. You pay per ride, no maintenance, no parking fees. But during rush hour, the meter runs fast — and sometimes you wait for one to show up.

The webhook handler is a Lambda — it runs maybe 100 times a day, so serverless is way cheaper than keeping a server running 24/7.

For PMs, this means serverless is great for low-traffic, event-driven workloads. But "serverless" doesn't mean free — at high volume, costs can exceed a traditional server. Ask about the expected traffic pattern before assuming it's the cheaper option.

IaC

Infrastructure as code

Infrastructure as code (IaC) means defining your cloud resources — servers, databases, networks, permissions — in code files (using tools like Terraform or Pulumi) instead of clicking through a cloud console. Changes are versioned, reviewed, and repeatable. To rebuild an environment, you run the code.

A recipe vs. cooking from memory. The recipe is written down, reproducible, and anyone can follow it. Cooking from memory works until you forget a step — or someone else needs to make the dish.

Don't create the database manually — add it to the Terraform config so it's tracked and reproducible.

For PMs, this means IaC is why environments can be spun up consistently and quickly. If your team doesn't use it, creating a new environment is manual, error-prone, and slow — which directly affects your ability to test and ship.

Load balancer

Distributing traffic so nothing gets overwhelmed

A load balancer sits in front of your servers and distributes incoming requests across multiple instances — so no single server gets crushed while others sit idle. If one instance goes down, the load balancer stops sending traffic to it. It's the reason your product stays responsive under heavy load.

A host at a busy restaurant. Instead of sending every party to the same table, the host spreads guests across open tables — and stops seating at any table that's full.

The API was timing out because all traffic was hitting one instance — we added a load balancer and response times dropped immediately.

For PMs, this means when engineers say "we need to add a load balancer," they're solving for reliability and performance under traffic. It's also why your product can have multiple servers running the same code — the load balancer decides which one handles each request.

FinOps

Managing and optimizing cloud costs

FinOps is the practice of managing cloud spending with the same rigor you'd apply to product metrics — tracking usage, rightsizing resources, catching waste, and making cost a shared responsibility between engineering, finance, and product. Cloud bills scale with usage, and without visibility, costs drift fast.

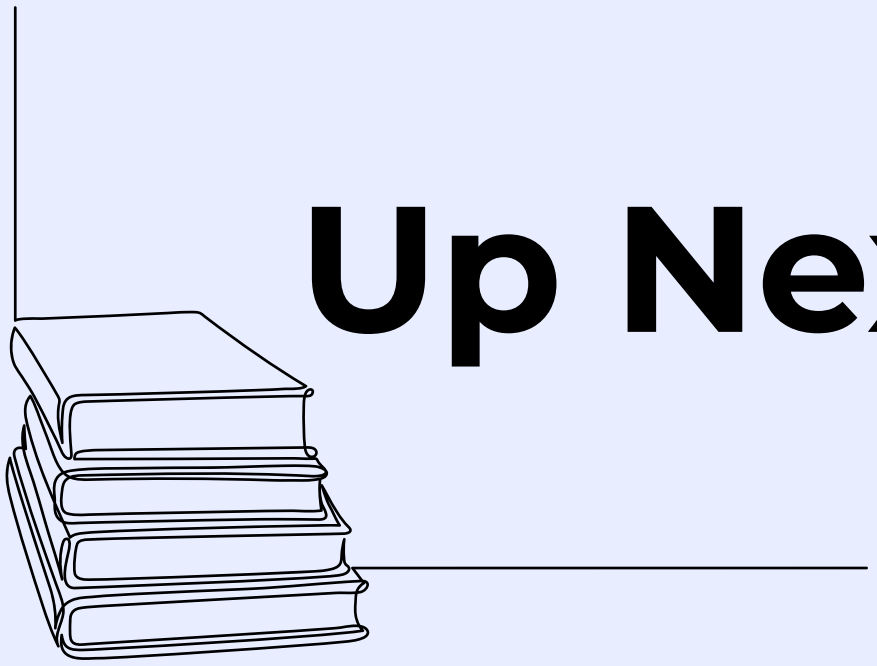
Tracking your subscriptions. Each one seems small — but without a monthly review, you're paying for three streaming services you forgot you had.

We're spending $12K/month on idle dev environments that nobody shuts down — that's an easy win.

For PMs, this means cloud costs are product decisions. Choosing a region, a caching strategy, or serverless vs. containers all affect the bill. If you're not involved in FinOps conversations, you're making cost decisions without knowing it.

Stay Tuned

Up Next



Your system is live. Users are showing up. But can it handle the load? Next up: a plain-English guide to scalability, availability, latency, and SLAs — the terms that explain what happens when your product meets the real world.