BUILD WITH IT

Shipping It

Get what you built in front of real people, and keep it alive.

First edition, September 2026

Addaly

Series editor: Dr Mustafa Mun

Draft, open for academic review

BUILD WITH IT

Shipping It

Get what you built in front of real people, and keep it alive.

Series editor: Dr Mustafa Mun

Addaly

First edition, September 2026 · Draft, open for academic review

Shipping It

First edition, September 2026, build 62a26075.

Written by AI models to a written standard, with Dr Mustafa Mun as series editor. Exam keys checked blind by a second model: 3,665 of 3,666 agreed. Academic review invited.

The manuscript was drafted with the assistance of a large language model and was edited and published under his name. That is stated plainly here rather than left to be discovered, because a reader is entitled to know how a book they are trusting was made.

How to cite: *Mun, M. (2026). Shipping It. Addaly. addaly.com/learn/cloud-and-shipping.*

This book is the printed form of the course of the same name at addaly.com/learn/cloud-and-shipping. The website is the living version and is corrected first. Where the two differ, the website is right.

Free to read, free to download, free to print, and free to teach from. There is no paid edition and there never will be.

Every diagram in it is drawn from data by code, not generated as a picture, so the numbers on the page are the numbers in the argument. The answers to every check, test and examination question are at the back.

8 modules · 74 lessons · 29 figures · 8 case studies · 56,931 words · about 11 hours

Brief contents

Contents 6

1 How the internet reaches your process 11

2 Making a deploy boring 57

3 Knowing what it is doing 107

4 Load, cost and the traffic that is not customers 151

5 Staying alive: the bad day and the day after 201

6 Shipping a model, not just a web app 247

7 Security after the short list 291

8 Running it for years 337

Examination 377

Answers 395

1

MODULE 1

How the internet reaches your process

Trace one request from a typed URL to your running process and back, and name which layer is at fault when it fails.

Before you can fix a live site you need a picture of what happens between somebody typing your name and your code running. Six layers, each with its own way of failing. This comes first because every later module assumes you can say which layer you are looking at.

1 What a server actually is 12
2 The first ten minutes on a new machine 15
3 What keeps your process running when you close the laptop 19
4 Why your site is not loading 23
5 What a status code is telling you 27
6 The padlock, and what it does not prove 30
7 What sits in front of your app 35
8 One request, with a stopwatch 39
9 The handshake you keep paying for 44
Case study: The mock test that resolved two ways 48
Module 1 test 54

Lesson 1 · 7 min

What a server actually is

THE ONE THING TO KEEP

A server is just a process holding a port open; the cloud only changes who owns the machine.

A server is a process, not a building

Run this on your laptop:

```
$ node server.js
Listening on http://localhost:3000
```

You are now running a server. A program is holding a TCP port open and answering whatever arrives on it. That is the whole definition. Nothing about a rack, a cooled room in Mumbai, or a company with a blue logo is part of it.

What your laptop lacks is not server-ness. It lacks three things: a public address (your router gave you a private one like 192.168.1.7), a certificate so browsers trust it, and the willingness to stay awake when you close the lid. Deploying is buying those three things from someone else.

What you are actually renting

There is a ladder, and the rungs differ in how much of the machine you are responsible for.

- **Bare metal.** A physical computer in a building. You get all of it, you patch the kernel, you care when a disk dies.
- **Virtual machine.** A slice of that computer, with its own Linux, its own IP, root access. A DigitalOcean droplet, an EC2 instance, a Hetzner box. Small ones run about $4–7 a month.
- **Container platform.** You hand over an image, the platform runs copies of it and restarts them when they die. Fly, Render, Cloud Run.

- **Serverless.** You hand over a function. It runs when a request arrives and stops afterwards. Lambda, Cloudflare Workers, Vercel functions.

"The cloud" means only this: someone else owns the machine, the building, the power and the network, and bills you by time or by request. It is a rental agreement, not a technology.

The axis that actually changes your code

Forget price for a moment. The question that changes how you write things is: **is a process always running?**

Always-on, you get a process with memory that persists. You can hold a database connection pool. You can keep an in-memory cache. You can start a timer that fires every ten minutes. You pay the same at 3am with zero visitors.

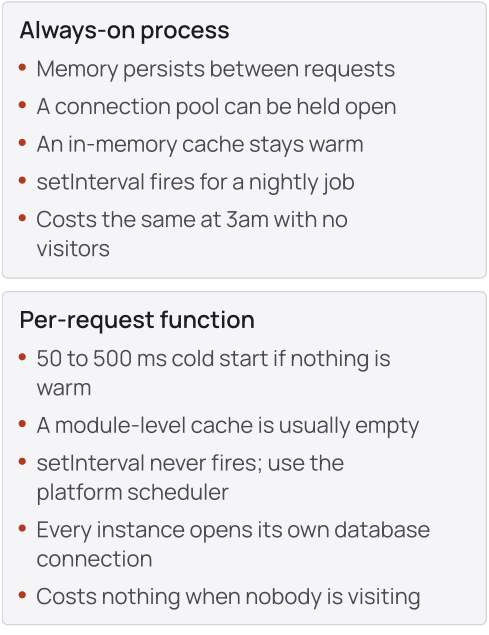
Per-request, a process starts when a request arrives — 50 to 500ms of cold start if none is warm — handles it, and may be gone before the next one. Consequences people discover the hard way:

- A module-level cache is usually empty, because the next request lands on a different instance.
- `setInterval` for a nightly job never fires. You need the platform's scheduler.
- Every instance opens its own database connection. Postgres defaults to about 100 connections. Fifty concurrent instances plus a pool of five each is 250, and you get `sorry, too many clients already`. This is why serverless projects put a connection pooler in front of the database.

Neither shape is better. They fail differently, and the failures are only surprising if you did not know which one you picked.

Figure 1.1

Always-on process versus per-request function



Neither shape is better. Each one fails in a way that is only a surprise if you did not know which one you picked.

The machine is in a place

Regions are not marketing. A round trip from Lagos to a Virginia region is roughly 110–180ms. Frankfurt might be 60ms. That is per round trip — if your handler runs ten sequential queries against a database on another continent, you have built a two-second page out of fast code.

The rule that matters more than any other: **put your application in the same region as your database.** Users are far away once. Your database is far away on every query.

So what is a deploy

Get the code onto a machine that has a public address, run it, point a name at it, keep it running. Everything else is convenience.

BEFORE YOU MOVE ON

Your handler stores computed results in a module-level object so repeat requests are fast. Locally, the second request is instant. Deployed to a per-request platform, the cache almost never hits — nearly every request recomputes. What is happening?

A. Requests are spread across instances that start fresh, so the object is usually empty for whichever one answers

B. The platform strips global state from deployed functions as a security measure

C. The cache is working, but network latency is large enough to hide the saved computation

D. The production build removed the cache object because nothing appeared to reference it

The answer and why: p. 397

Lesson 2 · 9 min

The first ten minutes on a new machine

THE ONE THING TO KEEP

A fresh machine is scanned within minutes, so keys-only login, a non-root user, a closed firewall and automatic security updates are not hardening you do later — they are the machine's first ten minutes.

A rented box is on the public internet within seconds

The moment a provider hands you an IP address, strangers start knocking. Put a fresh machine online with password login enabled and you will see failed SSH attempts within minutes — usually thousands a day, from botnets working through `root`, `admin`, `ubuntu`, `postgres` and every password in a leaked list. None of this is personal. It is a script.

So the first ten minutes on a new machine are the same ten minutes every time, and they are worth turning into a checklist you keep.

Keys, not passwords

Generate a key pair on your own machine, not on the server:

```
ssh-keygen -t ed25519 -C "laptop-2026"
ssh-copy-id -i ~/.ssh/id_ed25519.pub user@203.0.113.10
```

Ed25519 keys are short, fast, and the current default. RSA still works if you use 4096 bits, and you will meet it on older systems.

The private key never leaves your laptop. What you copy to the server is the `.pub` half, which lands in `~/.ssh/authorized_keys`. Anyone holding that file can let themselves in, which is why the permissions matter: `700` on `~/.ssh`, `600` on the file. OpenSSH silently refuses keys in a world-readable directory, and this is the single most common reason a key that "should work" does not.

Put a passphrase on the private key. Without one, a stolen laptop is a stolen server. `ssh-agent` means you type it once per session rather than once per connection.

Turn off what you are no longer using

In `/etc/ssh/sshd_config`:

```
PasswordAuthentication no
PermitRootLogin no
```

Then `sudo systemctl reload ssh`. Keep your current session open while you test the new one in a second terminal. Locking yourself out of a machine you have no console access to is a real way to lose a weekend, and it is entirely avoided by not closing the first window.

Changing the SSH port from 22 is popular advice. It removes most of the log noise and stops none of the serious scanning, which enumerates ports anyway. Treat it as tidying, not security.

Do not work as root

Create a normal user, give it `sudo`, and use it. Root has no undo and no audit trail: every process it starts, every file it writes, every `rm -rf` with a typo in it runs with the whole machine's authority. The habit costs nothing and it is the difference between a mistake and an outage.

Your application should run as its own user too, one with no login shell and no write access to its own code directory. If the process is ever made to run attacker-supplied code, that user's permissions are the blast radius.

Close the ports you are not using

Everything a fresh install starts — a database on 5432, Redis on 6379, an admin panel on 8080 — is reachable from the internet unless something stops it. Databases exposed this way are found and emptied within hours, and there are search engines dedicated to listing them.

```
sudo ufw default deny incoming
sudo ufw allow 22,80,443/tcp
sudo ufw enable
ss -tlnp    # what is actually listening, and on which address
```

Read the `ss` output carefully. `0.0.0.0:5432` means the whole internet; `127.0.0.1:5432` means only this machine. A cloud provider's security group does the same job one layer out, and it is fine to have both.

Patches, applied by something other than your memory

On Debian or Ubuntu:

```
sudo apt install unattended-upgrades
sudo dpkg-reconfigure -plow unattended-upgrades
```

This installs security updates on its own. It will occasionally need a reboot to take effect, which is what `/var/run/reboot-required` is telling you. Unattended upgrades are the highest-value five minutes on this list, because the alternative is a machine that stays on a kernel with a public exploit until you happen to think about it.

Two habits that save you later

- **Write the machine down.** One file in your repository: what it is, who pays for it, what runs on it, which DNS names point at it. A server nobody can name is a server nobody dares turn off.
- **Assume it is disposable.** If rebuilding this box from nothing would take you more than an hour of remembering, the recipe should be in a script, not in your head. That is the whole argument for configuration management, and a shell script committed to the repository already counts.

The free path

All of this is OpenSSH, `ufw` and `systemd`, which ship with every Linux distribution and cost nothing. `mosh` is worth installing if your connection drops on trains. Managed platforms hide this entire lesson from you, which is a genuine advantage — until the day you need to know what a machine is doing, and the machinery underneath is the same either way.

BEFORE YOU MOVE ON

You copy your public key to a new server, then set `PasswordAuthentication no`. Your next SSH attempt is refused with "Permission denied (publickey)", although the same key works on another server. What is the most likely cause?

A. The key is Ed25519 and the server only accepts RSA, so it was never offered

B. The firewall is blocking port 22, so the connection never reaches sshd

C. `PasswordAuthentication no` also disables public-key authentication until `PubkeyAuthentication yes` is added explicitly

D. Permissions on `~/.ssh` or `authorized_keys` are too open, so sshd is ignoring the file

The answer and why: p. 398

Lesson 3 · 9 min

What keeps your process running when you close the laptop

THE ONE THING TO KEEP

Something other than your terminal has to own the process: a supervisor that starts it at boot and restarts it on exit, plus the journal that says why it exited — and a process that keeps vanishing without a log line was probably killed by the kernel for using too much memory.

The thing people do first, and why it fails

The first deployment nearly everybody performs is this: SSH in, run `npm start` or `python app.py`, see the site working, close the terminal. An hour later the site is down.

CASE STUDY · MODULE 1

The mock test that resolved two ways

An illustrative case, built from documented patterns.

A coaching centre in Kota moves its test-series site to a bigger machine the night before four thousand students sit a Sunday mock paper.

Sharma Classes runs a weekly test series for about four thousand students. The papers are taken on phones, in a two-hour window on Sunday morning, on a site the centre's one part-time developer built and keeps running on a rented virtual machine.

The old machine was small and had fallen over twice during a paper. So on Saturday evening the developer built a bigger one, copied the application across, restored the database, checked the site by typing the new IP address into his browser, and at half past ten at night changed the A record for `tests.sharmaclasses.in` from the old address to the new one.

At six on Sunday morning the phone started ringing.

Some students were seeing the new site, with Sunday's paper on it. Others were seeing the old site, which was still running, still showed last week's paper, and had a database nothing was writing to any more. A few were getting a full-page browser security warning. Nobody could work out what separated the three groups, because the same student got a different answer on the hostel wifi and on mobile data.

Three facts were established in ten minutes, with one command each:

- `dig +short tests.sharmaclasses.in @ns1.provider.net` — the authoritative nameserver returned the new address. The change had saved.
- `dig +short tests.sharmaclasses.in @1.1.1.1` — a public resolver returned the old address. It had asked before the change and was still holding the answer.
- `dig tests.sharmaclasses.in` — the record's TTL was 86400. One day. Nobody had touched it, because it was the default the provider set.

Nothing propagates. Records are pulled and then cached, and every cache holds its copy until the TTL runs out. A resolver that asked at nine on Saturday evening would keep handing out the old address until nine on Sunday evening — the whole of the test window and most of the evening after it. Lowering the TTL at six in the morning changes nothing for a cache that is already holding a copy, because the copy it holds carries the old number with it.

The security warning turned out to be a different fault wearing a DNS costume. The certificate on the new machine had been issued for `tests.sharmaclasses.in` and nothing else. Students who typed `www.tests.sharmaclasses.in`, which the old machine had also answered for two years, were meeting a name the new certificate did not cover.

So at ten past six, with the paper opening at nine, there were two options and a real cost on each.

Turn the old machine off. Then every student still resolving to the old address gets nothing at all — a connection refused, for some of them until that evening. It is clean, it is honest, and it would have stopped a large share of four thousand students from sitting a paper they had paid for.

Leave the old machine running and make it forward. Put a reverse proxy on the old box that passes every request through to the new server, so both addresses serve the same site and the same database. The costs are not zero: an extra network hop for everyone still on the old address, a second machine to pay for and to remember to turn off, and a certificate on the old box that has to be valid for every name it answers to — including the `www` name that was already broken.

The centre's owner asked a third question, which was whether to postpone the paper by a week. Four thousand students had planned a specific Sunday around it, several had travelled, and a postponement is not free either.

YOUR ANSWERS

1. Why would lowering the TTL at six on Sunday morning not have helped anybody who was already seeing the old site?

2. The old machine served last week's paper from a database nothing was writing to. What is the more dangerous version of that situation, and how would you avoid it?

3. Students on wifi and on mobile data saw different sites. What does that tell you before you run a single command?

STOP HERE

Write your answers before you turn the page. How it played out starts on p. 52.

This page is blank so that how it played out stays out of view until you turn the page.

CASE STUDY · MODULE 1

How it played out

The developer kept the old machine alive and put Caddy in front of it, forwarding to the new server, with a certificate covering both the plain name and the `www` name. That took about twenty minutes and was in place before seven. The paper ran at nine with no further complaints, and nobody was told anything had happened.

The old machine stayed up for six days. It was retired only after `dig +short ... @1.1.1.1` had agreed with the authoritative answer for a full day and its access log had been silent for twenty-four hours.

The centre changed three things afterwards. The TTL on every record is lowered to sixty seconds two days before any planned change and raised again a day after it. Certificates are checked with `openssl s_client` for every name the site answers to, `www` included, and a scheduled job alerts at twenty-one days remaining, because Let's Encrypt no longer sends the warning email. And no infrastructure change is made in the twelve hours before a test — a rule written on the office wall rather than in a document nobody opens.

The questions, discussed

1. Why would lowering the TTL at six on Sunday morning not have helped anybody who was already seeing the old site?

A TTL travels with the answer. A resolver that fetched the record on Saturday evening received it stamped with 86400 seconds and will not ask again until that runs out, whatever the authoritative server now says. Lowering the TTL only affects answers handed out after the change, which is exactly why it has to be done days in advance rather than at the moment you need it.

2. The old machine served last week's paper from a database nothing was writing to. What is the more dangerous version of that situation, and how would you avoid it?

The dangerous version is an old machine that is still writing — pointed at its own copy of the database, taking answers from the students who still resolve to it. You then have two live databases, both correct, both incomplete, and no

clean way to merge them. Avoid it by making the old address stop being a destination and start being a forwarder: proxy every request to the new server so there is exactly one database, or take the old machine down entirely and accept the outage.

3. Students on wifi and on mobile data saw different sites. What does that tell you before you run a single command?

That the two devices are asking different resolvers and getting different answers, so the fault is in name resolution or its caching rather than in the server, the application or the certificate. One machine cannot serve two different sites to two people at the same instant; two addresses can. That single observation narrows the search to DNS before any diagnosis begins.

MODULE 1 TEST

Check what stuck

6 questions, one right answer each. This one is for you, not for a record. The answers, and the reason behind each, are in the key on p. 396. If two go wrong, re-read the module before the exam.

1 You changed an A record an hour ago. You see the new server; a colleague still sees the old one. What settles where the problem is, fastest?

A. Open the registrar's control panel and confirm the record was saved correctly

B. Ask the authoritative nameserver and a public resolver the same question

C. Lower the TTL to sixty seconds and wait for the change to spread

D. Clear both browsers and reload the page with the cache disabled

2 A signed-in user requests a page belonging to another team. Your handler answers 401. What does that cause?

A. The CDN caches the refusal and serves it to other users for a minute

B. Their client tries to authenticate again, against a login they already completed

C. The browser retries with the same cookie until the request times out

D. Search engines drop the page from their index as unavailable

3 Your renewal cron ran at day sixty and reported success, but visitors still get the expired certificate. What has most likely happened?

A. HSTS is holding the old certificate in the browser until max-age expires

B. The issuer rate-limited you and silently returned the previous certificate

C. The new certificate is on disk and the web server was never reloaded

D. The server clock is wrong, so the new certificate is not yet valid

4 You turn on your framework's trust-proxy setting and rate-limit on the leftmost entry of X-Forwarded-For. What have you built?

A. A limiter that treats every visitor behind the proxy as one client
B. A limiter that blocks your own CDN's addresses under load
C. Nothing new, because the proxy overwrites the header on every request
D. A limiter anybody can step around by sending the header themselves

5 Curl reports a 661 ms request from Mumbai to a server in Virginia, of which your handler accounts for 236 ms. Which change removes the most time?

A. Serve the response from a location nearer the user
B. Double the CPU and memory on the instance
C. Add an index to the slowest query in the handler
D. Enable compression on the response body

6 Behind a load balancer you get a small, steady drizzle of 502s that nothing in your application logs explains. Which mismatch produces exactly that?

A. The application is listening on 127.0.0.1 rather than on all interfaces
B. The balancer sends Connection: close, so every request opens a socket
C. The application's idle keep-alive timeout is shorter than the balancer's
D. The application's request timeout is shorter than the balancer's read timeout

on a Tuesday.

Containers, briefly and honestly

A container image is a filesystem plus a command. It solves one problem well: the machine's own state — system libraries, locales, that `imagemagick` you installed in 2023 — stops being part of your program.

It is not free. You now maintain a Dockerfile, and image builds have their own failure modes. For a static site or a simple app on a managed platform, you may never need one. For anything with native dependencies, it earns its keep quickly.

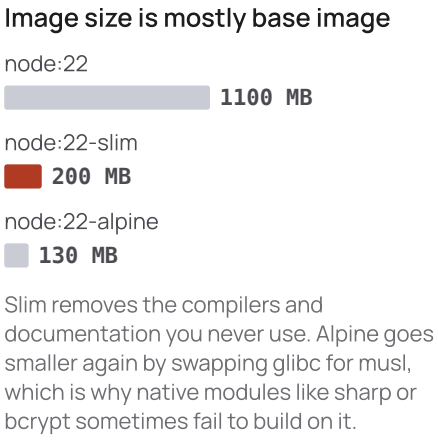
Four things worth knowing before you write one:

Layer order decides your build time. Each instruction is a cached layer, invalidated by the first change. Copy your lockfile and install dependencies *before* copying your source, or every one-character edit reinstalls the world.

```
FROM node:22-slim
WORKDIR /app
COPY package.json package-lock.json ./
RUN npm ci --omit=dev
COPY . .
CMD ["node", "server.js"]
```

Image size is mostly base image. `node:22` is around 1.1 GB. `node:22-slim` is around 200 MB. `node:22-alpine` is around 130 MB — and Alpine uses musl instead of glibc, so native modules like `sharp` or `bcrypt` may fail to build, or build and then behave differently. Use `-slim` unless you have measured a reason.

Figure 2.1



A `.dockerignore` is not optional. Without one you copy `node_modules`, `.git` and your `.env` into the image. The last one is a leaked secret sitting in a layer that people can pull.

Architecture is a real trap. Building on Apple Silicon produces an arm64 image. Deploying that to an x86 server gives `exec format error`, which reads like a corrupt binary. Build with `--platform linux/amd64`, or build in CI, which is x86 by default.

Docker Desktop needs a paid licence at larger companies. The free paths are identical in practice: Podman, Colima, Rancher Desktop, or the plain Docker engine on any Linux box.

Build in CI, not on your laptop

A laptop accumulates. Global installs, an old cache, an environment variable exported months ago in a shell profile, an authentication token that only exists there. CI starts from nothing every time, which is precisely why it catches things you cannot.

The one that stings is case sensitivity. macOS filesystems are case-insensitive by default; Linux is not. `import Button from "./button"` works forever on your Mac and fails in production with a bare module-not-found. There is no warning locally, ever. A Linux CI build finds it the first time.

EXAMINATION

Shipping It — final examination

This paper covers the whole course:

- how a request reaches your process
- making a deploy boring and reversible
- knowing what your system is doing
- load and cost and the traffic that is not customers
- staying alive on the bad day and the day after
- shipping a model rather than a web app
- security past the short list
- running the thing for years

56 questions, the whole pool. The site draws 25 for a sitting, pass mark 70%. Work any 25 under your own conditions. Answers from page 447.

1 Module 1 · How the internet reaches your process

You move an always-on application to a per-request serverless platform. Which of your existing habits silently stops working?

A. Reading configuration from environment variables at start-up
B. A nightly job scheduled with setInterval inside the process
C. Writing one structured log line per request to stdout
D. Returning a JSON body with an explicit status code

2 Module 1 · How the internet reaches your process

On a new machine, `ss -tlnp` shows Postgres listening on `0.0.0.0:5432`. What have you just learned?

A. Postgres is reachable from the whole internet unless something else stops it
B. Postgres is bound to the loopback interface and is safe from outside
C. The port is open locally but blocked by the default firewall policy
D. Postgres is running as root and should be moved to its own user

3 Module 1 · How the internet reaches your process

Your process keeps vanishing. There is nothing in its own logs before each disappearance — no exception, no shutdown line. Where do you look?

A. The supervisor's restart-limit settings, which suppress the final error
B. The proxy's error log, which records upstream connection failures
C. The kernel log, for an out-of-memory kill
D. The deploy history, for a release that overlapped the crash

4 Module 1 · How the internet reaches your process

Why do hosting instructions always say to use a CNAME for `www` and an A record for the bare domain?

A. A CNAME cannot point at an address outside your own zone
B. The specification forbids a CNAME beside other records at the apex
C. Apex records are answered by the registrar rather than the nameservers
D. CNAME lookups add a second round trip that the apex cannot afford

5 Module 1 · How the internet reaches your process

Why must a destructive action never sit behind a plain link such as `/posts/12/delete`?

A. Browsers cache GET responses, so the deletion appears not to happen

B. Search engines penalise sites whose links change server state

C. The URL is visible in logs, which leaks the identifier being deleted

D. Prefetchers, crawlers and link scanners will follow it with no human involved

6 Module 1 · How the internet reaches your process

You ship `Strict-Transport-Security: max-age=31536000; includeSubDomains` and one of your subdomains is HTTP-only. What is the position?

A. That subdomain is unreachable, and you cannot reach the browsers to undo it

B. That subdomain is unreachable until you remove the header from responses

C. Only browsers that visit the subdomain directly are affected by the policy

D. Nothing breaks, because the policy applies to the exact host that sent it

7 Module 1 · How the internet reaches your process

Your service creates a fresh HTTP client for every outbound call to a payment API. What does that cost under load?

A. Extra memory, because each client allocates its own buffers

B. Nothing measurable, since connection setup happens in the background

C. A TCP and TLS handshake per call, and sockets piling up in TIME_WAIT

D. Rate limiting at the provider, which counts connections rather than requests

Module 1 · How the internet reaches your process

The module starts on p. 11.

MODULE 1 TEST

Module 1 test, Q1, p. 54
You changed an A record an hour ago. You see the new server; a colleague still sees the old one. What settles where the problem…
Answer B: Ask the authoritative nameserver and a public resolver the same question
Why: The two answers separate the two possible faults. If the authoritative server returns the new address and a public resolver returns the old one, the change saved and you are waiting on a cached copy to expire. If the authoritative server returns the old address, the change never took effect there — often because the nameservers are somewhere other than the registrar whose form you edited.

Module 1 test, Q2, p. 54
A signed-in user requests a page belonging to another team. Your handler answers 401. What does that cause?
Answer B: Their client tries to authenticate again, against a login they already completed
Why: 401 means "I do not know who you are, send credentials", so a client that receives it does the only sensible thing and starts an authentication flow — for somebody who is already authenticated, that is a loop. The honest code is 403: I know exactly who you are, and no.

Module 1 test, Q3, p. 54
Your renewal cron ran at day sixty and reported success, but visitors still get the expired certificate. What has most likely happened?
Answer C: The new certificate is on disk and the web server was never reloaded
Why: A web server reads the certificate at start-up and keeps it in memory. Renewal writes a new file; nothing tells the running process. The renewal hook has to reload the server, and the user running the cron needs permission to do it. Clock skew and rate limits are real failures, but they stop issuance rather than leaving a fresh file unserved.

Module 1 test, Q4, p. 55
You turn on your framework's trust-proxy setting and rate-limit on the leftmost entry of X-Forwarded-For. What have you built?
Answer D: A limiter anybody can step around by sending the header themselves
Why: X-Forwarded-For is an ordinary header that a client can set on its first request, and each hop appends to the list. The leftmost entry is therefore whatever the caller typed. Only an entry appended by a proxy you control is trustworthy, which in practice means reading from the right, or using a header your CDN overwrites rather than appends.

Module 1 test, Q5, p. 55

Curl reports a 661 ms request from Mumbai to a server in Virginia, of which your handler accounts for 236 ms. Which change removes the…

Answer A: Serve the response from a location nearer the user

Why: Roughly four hundred milliseconds of that request was DNS, the TCP handshake and the TLS handshake — round trips across thirteen thousand kilometres, which no amount of server tuning touches. Shortening the distance is the only lever that reaches them. Making a 236 ms handler faster is worth doing later, and it cannot win here.

Module 1 test, Q6, p. 55

Behind a load balancer you get a small, steady drizzle of 502s that nothing in your application logs explains. Which mismatch produces exactly that?

Answer C: The application's idle keep-alive timeout is shorter than the balancer's

Why: If the balancer believes a pooled connection is still usable for longer than your application does, the application will close one at the moment a request is being handed to it. That request fails with nothing on your side to log. Making the application's idle timeout the longer of the two removes it. A wrong bind address fails every request rather than a few; Connection: close costs handshakes, not errors.

THE LESSON CHECKS

Lesson 1, p. 15

Your handler stores computed results in a module-level object so repeat requests are fast. Locally, the second request is instant. Deployed to a per-request platform…

Answer A: Requests are spread across instances that start fresh, so the object is usually empty for whichever one answers

Why: A server is just a process holding a port open; the cloud only changes who owns the machine.

B The platform strips global state from…: Assumes the platform sandboxes away a language feature, rather than simply not keeping any one process alive between requests.

C The cache is working, but network…: Assumes latency dominates everything, so an in-process win could not show up in the timings — yet locally the same code showed a clear difference.

D The production build removed the cache…: Assumes a behaviour difference between dev and production must come from the build step, which is often true but not here.

Lesson 2, p. 19

You copy your public key to a new server, then set `PasswordAuthentication no`. Your next SSH attempt is refused with "Permission denied (publickey)", although the…

Answer D: Permissions on `~/.ssh` or `authorized_keys` are too open, so sshd is ignoring the file

Why: A fresh machine is scanned within minutes, so keys-only login, a non-root user, a closed firewall and automatic security updates are not hardening you do later — they are the machine's first ten minutes.

A The key is Ed25519 and the…: Assumes modern OpenSSH rejects Ed25519. It has been the recommended default for years; algorithm mismatch is a real failure, but it is rare and would show as a specific unsupported-type error.

B The firewall is blocking port 22…: Confuses a network failure with an authentication failure. A blocked port gives a timeout or connection-refused, not a "Permission denied" from the server.

C `PasswordAuthentication no` also disables public-key authentication…: Treats the two settings as coupled. Public-key authentication is on by default; the password setting only removes the password path.

Lesson 3, p. 23

Your service disappears every few days. There is nothing in the application log, no stack trace, and no error at all — the log simply…

Answer D: The kernel's OOM killer terminated the process, which gets no catchable signal and so writes nothing

Why: Something other than your terminal has to own the process: a supervisor that starts it at boot and restarts it on exit, plus the journal that says why it exited — and a process that keeps vanishing without a log line was probably killed by the kernel for using too much memory.

A An unhandled exception is crashing the…: Plausible, but an unhandled exception normally prints a stack trace to stderr, which the journal captures even when the application's own logger does not.

B The SSH session that started it…: The classic beginner failure, but it happens once, at the moment the terminal closes, not repeatedly on a service started by a supervisor.

C systemd hit its start rate limit…: Describes what happens after repeated fast restarts, not a cause of the first disappearance — and it would leave the service down, not running again.

BUILD WITH IT

Shipping It

Get what you built in front of real people, and keep it alive.

WHAT IT TEACHES	Most of what stops people shipping is not code. It is a domain that will not resolve, a key committed by accident, a bill that arrives three weeks later, and a backup nobody has ever restored.
WHAT IS INSIDE	Eight modules and 74 lessons, 29 figures drawn from data, eight case studies, eight module tests, a 56-question examination, and every answer with the reason behind it.
LICENCE	CC BY-NC-SA 4.0. Copy, print, share and adapt it for any non-commercial purpose, with credit, under the same licence. There is no paid edition.
THE COURSE	Free to read, with the lessons, the films and the examination, at **addaly.com/learn/cloud-and-shipping**

Scan to open the course

Addaly

First edition, September 2026 · build 62a26075