XXVII

Monty Hall od nuly do produkce

Vyvrcholení. Sedneš k prázdnému adresáři a reálnými nástroji roku 2026 postavíš veřejnou službu: spec, agent ji napíše, testy ji ohlídají, git ji zverzuje, kontejner ji zabalí, linka ji nasadí — a na živém dashboardu se před očima usadí 66,7 %. Matematika z kapitoly čtrnáct, produkce z kapitoly jednadvacet. Tohle je klenák celého dějství inženýr: složení všeho, cos v něm osahal, do jedné běžící věci.

Jak nazvat ten druhý konec spektra, kde ostřílení profesionálové zrychlují svou práci jazykovými modely, a přitom hrdě a sebejistě ručí za software, který vyrobí?

— Simon Willison, Vibe engineering, *simonwillison.net, 7. 10. 2025*

Prázdný adresář v pátek večer

Je pátek večer a před tebou je to nejlevnější, co v téhle profesi existuje: prázdný adresář. Jedna složka, žádný soubor. `mkdir monty && cd monty` — a kurzor bliká v prázdnu. Přesně sem se vešlo dvacet let bolesti z první části téhle knihy; a přesně odsud dnes odejdeš za jeden večer s veřejnou službou, na kterou se může připojit kdokoli na světě.

Rozhodl ses postavit tu jednu věc, kterou celá kniha slibovala jako svůj klenák: simulátor tří dveří jako běžící produkci. Ne applet, co žije jen na tvém notebooku, dokud se díváš — skutečnou službu s dashboardem, na kterém se návštěvníkům před očima usazuje podíl výher strategie „měním“ na dvou třetinách. Ten samý rozhodčí, který v kapitole čtrnáct smířil Erdőse a tisícovku doktorů — teď běžící pro celý svět, i když u toho nikdo nekouká.

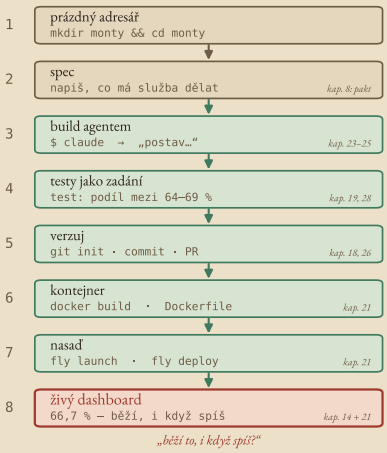
Před pěti lety by to byla práce na týden a znalost tuctu nástrojů, které se člověk učil roky. Dnes máš po ruce agenta, který umí psát kód, spouštět příkazy a číst chybové hlášky. Ale pozor — a tohle je celá pointa dějství inženýr: agent ti nenahradí úsudek, jen syntax. Pořád musíš vědět, *co* chceš postavit, *jak* poznáš, že to funguje,

a *kdy* to nesmí ven. Tvoje práce se přesunula z psaní řádků do rozhodování, které řádky mají vzniknout a jak se ověří. Kentaur z kapitoly jedna, teď v jednom večeru u jednoho adresáře.

Willisonův citát v čele není náhoda. On sám razí rozdíl mezi *vibe codingem* — „nezodpovědným stavěním softwaru hodem kostkou, kdy je jedno, jaký kód vznikne“ — a *vibe engineeringem*: totéž zrychlení modelem, ale s plnou odpovědností za výsledek. Tahle kapitola je celá o tom druhém. Postavíme rychle a s pomocí agenta; ale za každý řádek ručíme testem, verzí a bránou. To je rozdíl mezi hračkou, která měla štěstí, a produkcí, která běží, když spíš.

Celý oblouk na jeden pohled

Než sáhneš na klávesnici, podívej se na celou cestu shora. Osm zastávek, jedním směrem — od prázdného adresáře k dashboardu, na kterém svítí 66,7 %. Každá zastávka je jedno řemeslo, které jsi v tomhle dějství osahal zvlášť; capstone je spojí do jedné linky.



Hrdinský graf kapitoly: celý oblouk jedné služby reálnými nástroji. Prázdný adresář (1) → spec (2) → agent staví (3) → testy jako zadání (4) → verzuj (5) → kontejner (6) → nasaď (7) → živý dashboard (8). U každé zastávky skutečný příkaz a spojnice na kapitolu, odkud řemeslo pochází. Osmá, sanguine zastávka je klenák: dashboard, na němž se usazuje 66,7 %. Šipky vedou jedním směrem — to je linka *z kap. 21, ne roura z kap. 17.*

> *Všimni si, že sedm z osmi zastávek nemá nic společného s tím, že služba počítá zrovna tři dveře. Prázdný adresář, spec, testy, git, kontejner, deploy, monitoring — to je* tatáž *linka pro appku na počasí, pro interní nástroj i pro věc, co poletí na milion lidí. Monty Hall je jen náklad. Kouzlo, jak víš od McLeana, není v nákladu; je v tom, že bedna má všude stejné rohy.*

Projdeme tu cestu shora dolů. Batole dostane celý příběh souvisle; teenager u každé zastávky konkrétní příkaz; Einstein na konci matematiku, proč se ten dashboard vůbec smí usadit. Jdeme.

Batole: jeden večer, osm kroků

Sedneš k prázdnému adresáři a řekneš nahlas, co chceš: „webová služba, kde si návštěvník zahraje tři dveře, a dashboard, na kterém běží podíl výher strategie „měním“ ke dvěma třetinám.“ To je *spec* — zadání. Ještě jsi nenapsal řádek kódu, ale už jsi udělal nejtěžší část: rozhodl jsi, *co* stavíš a *jak* poznáš, že je to hotové.

Pak spustíš agenta — napíšeš `claude` a řekneš mu to zadání vlastními slovy. Agent začne psát: založí projekt, napíše logiku hry, přidá stránku s dashboardem, rozběhne to u tebe na notebooku. Za pár minut běží první verze. Ty ji vyzkoušíš, klikneš pár her — funguje. Tady většina lidí skončí a řekne „hotovo“. Ty ne, protože jsi četl první část téhle knihy: běží to jen proto, že se *díváš*. To je pořád hračka.

Takže uděláš to, co dělá inženýr, ne nadšenec: napíšeš (nebo necháš napsat) *testy* — a napíšeš je jako *zadání*, ne jako dodatečnou kontrolu. Řekneš: „po deseti tisících hrách musí podíl výher „měním“ ležet mezi 64 a 69 procenty.“ To je invariant — věc, která musí platit vždycky, ať se v kódu změní cokoli. Když ho agent poruší, test zčervená a ty víš, že se něco rozbilo, dřív, než to uvidí kdokoli jiný.

Teď to *zverzuješ*: `git init`, první commit. Odteď má každá změna svůj checkpoint a každý experiment vlastní větev — máš undo pro celé odpoledne (kapitola osmnáct) a pustíš agenta z řetězu beze strachu, protože se vždycky můžeš vrátit. Pak službu *zabalíš do kontejneru* — bedny, která poběží stejně u tebe i na cizím serveru — a pošleš ji *linkou* na veřejný server. Linka má *bránu*: bez prošlých testů nepustí do produkce nic.

A je to. Otevřeš prohlížeč, zadáš veřejnou adresu — a tam běží tvůj dashboard. Zavřeš notebook, jdeš spát. Ráno se podíváš: přes noc si zahrálo pár lidí, číslo se drží u 66,7 %, služba o sobě dala vědět, že žije. *Postavil jsi produkci.* Ne v myšlenkách; doopravdy, za jeden večer, u prázdného adresáře.

> A tady je ta dobrá zpráva, kvůli které za celé dějství stálo: nasadit takovou službu dneska nestojí přístav, jeřáb ani McLeanovu odvahu. Stojí to večer, jeden agent za pár dolarů tokenů a server za necelé dva dolary měsíčně. Logistiku, o jaké se roku 1956 nesnilo ani námořním velmocem, máš ty sám na stole — a z velké části zadarmo. Píseň je skutečně krásná.

Teenager: každá zastávka reálným příkazem

Teď to samé rukama, s příkazy, které opravdu napíšeš. Datováno k červenci 2026; nástroje se mění, princip u každé zastávky ne.

1–2 · Prázdný adresář a spec

TEENAGER · MECHANISMUS

Nezačínáš kódem, začínáš *zadáním*. Spec je pakt z kapitoly osm: napíšeš dřív, co má platit, než tě pokušení („však to nějak dopadne“) dostane.

```
$ mkdir monty && cd monty
$ $EDITOR SPEC.md          # zadání vlastními
slovy, ne kód:
```

```
# Monty — veřejná služba
- Návštěvník zahraje hru o tři dveře, strategie
„měním“.
- Dashboard: běžící podíl výher, cíl 2/3, pásmo
±2·SE.
- INVARIANT: po 10 000 hrách je podíl výher mezi
64 a 69 %.
- Když spadne databáze, služba nepadá — degraduje
se s grácií.
```

Ten `SPEC.md` je zároveň nejlepší prompt pro agenta *i* zadání pro testy. Jeden soubor, dvě role: řekne stroji, co stavět, a tobě, jak poznáš, že se ti nepodsouvá nesmysl. Píšeš ho ty, ne model — tohle je ten úsudek, který ti agent nevezme.

→ kap. 8: pakt = smlouva s budoucím já. Spec zapsaný před *generováním je přesně on: až agent vyrobí něco, co „skoro funguje“, spec je to, oč se opřeš, když říkáš „ne, tohle ještě není hotové“.*

→ kap. 25: dobrý spec je i skill: řemeslo předané modelu textem. Čím přesněji napíšeš, co *chceš, tím méně budeš opravovat,* co *vyrobil.*

3 · Build agentem

TEENAGER · MECHANISMUS

Spustíš agenta v terminálu a dáš mu spec. On píše kód, spouští příkazy, čte chyby a opravuje se — ty ho vedeš a kontroluješ.

```
$ claude
> Přečti SPEC.md a postav tu službu. Použij malý
web
  framework, logiku hry drž oddělenou od webu, ať
jde
  testovat samostatně. Po každém kroku mi ukaž,
cos udělal.
```

Klíč je poslední věta. Nenech agenta zmizet na deset minut a vysypat tisíc řádků, které nikdo nečetl — to je vibe coding hodem kostkou. Nech ho pracovat po krocích a *čti, co dělá*. Logiku hry (losování dveří, strategie „měním“) si necháš oddělit od webové slupky právě proto, abys ji mohl otestovat izolovaně — pyramida testů z kapitoly devatenáct začíná u návrhu, ne až u testů.

→ kap. 23–24: agent v terminálu (Claude Code) vs. IDE (Antigravity), sandbox a schvalování. Pro capstone stačí terminálový agent v adresáři projektu; nebezpečné příkazy (mazání, síť) drž za schválením — Replit, který smazal produkci (kap. 24), je varování, ne legenda.

4 · Testy jako zadání

TEENAGER · MECHANISMUS

Nejdřív *hloupý soupeř* (kap. 12): strategie „nechávám“. Když ji tvůj kód neporazí, něco je špatně. Pak invariant ze specu jako opravdový test:

```
# test_monty.py
def test_menim_porazi_hloupeho_soupere():
    p_menim   = simuluj(strategie="měním",
n=10_000)
    p_nechavam = simuluj(strategie="nechávám",
n=10_000)
    assert p_menim > p_nechavam        # kentauří
kontrola

def test_invariant_dve_tretiny():
    p = simuluj(strategie="měním", n=10_000)
    assert 0.64 < p < 0.69             # spec,
černé na bílém
```

```
$ pytest -q
..                                          [100%]
```

Tyhle dva testy jsou tvoje *specifikace pro agenta*: napsal jsi je sám a necháš stroj plnit je. Obrácení rolí z kapitoly devatenáct — úsudek zůstává tobě, dřinu dělá on. Kdyby agent „omylem“ napsal moderátora, který otvírá náhodně (Monty Fall z kap. 14!), `test_invariant` spadne na padesáti procentech a chytí to dřív, než to uvidí návštěvník.

→ kap. 19: testy jako zadání pro AI — napiš je sám, nech stroj plnit. → kap. 28 (dějství vědec v praxi): jak poznáš, že to agent *napsal dobře — evals, regrese, hloupý soupeř i tady. Test je nejlevnější eval, jaký máš.*

→ kap. 14: `test_invariant` *je refrén té kapitoly zabetonovaný do stroje: „měním“ musí vyjít kolem 2/3; když ne, buď máš bug, nebo špatný protokol.*

5 · Verzuj

TEENAGER · MECHANISMUS

Teprve teď — když testy prošly — commituješ. Rituál z kapitoly osmnáct: commit před každým „přepiš to celé“, větev na každý experiment.

```
$ git init
$ git add -A
$ git commit -m "Monty jako služba: hra,
dashboard, testy zelené"
$ git switch -c experiment-rychlejsi-dashboard
# větev = paralelní vesmír
```

Když pustíš agenta na větev a on to rozbije, `git switch main` tě vrátí do bezpečí — undo pro celé odpoledne. A než cokoli pošleš do produkce, přečteš *diff* vlastníma očima (nebo si ho necháš zrevidovat), protože AI výstup je předpověď, ne pravda: revize diffu je poslední brána, kterou drží člověk, ne stroj.

→ kap. 26: git v praxi s agenty — worktrees, commit před experimentem, `git diff` *a PR review AI výstupů. Zlaté pravidlo dějství:* každý tah proti verzovacímu systému. *Agent smí zkoušet cokoli, dokud existuje* `git switch main`.

TEENAGER · MECHANISMUS

Bedna z kapitoly jednadvacet, teď doopravdy. `Dockerfile` je McLeanův recept: čtyři věty, čtyři rohy — a „u mě to funguje“ přestane být výmluva.

```
# Dockerfile
FROM python:3.12-slim          # ZÁKLAD: hotový
podklad s jazykem
WORKDIR /app
COPY . /app                    # ZKOPÍRUJ: můj
kód dovnitř
RUN pip install -r requirements.txt   # PŘIPRAV:
jednou, při balení
CMD ["python", "-m", "monty.web"]     # SPUSŤ:
pokaždé, při startu
```

```
$ docker build -t monty .
$ docker run -p 8080:8080 monty   # běží u tebe
přesně jako poběží venku
```

`RUN` se odehraje *jednou*, když bednu balíš; `CMD` *pokaždé*, když ji někdo zapne. To je celé kouzlo přenositelnosti: co by se lišilo stroj od stroje, je zapečetěné uvnitř. A bedna má být malá a bez tajemství — hesla nikdy do image, ta se předají až za běhu.

→ kap. 21: kontejner a jeřáb — kouzlo je v rozích, ne v krabici. `FROM`, `COPY`, `RUN`, `CMD` *jsou ty čtyři rohy. Nástroj (Docker, Podman) je logo; princip „zabal se vším a zapečeť“ je starší i širší.*

TEENAGER · MECHANISMUS

Bednu pošleš na veřejný server. Ať už poskytovatelem, který umí vzít `Dockerfile` přímo, dvěma příkazy:

```
$ fly launch      # jednou: založí appku, vygeneruje konfiguraci
$ fly deploy      # postaví image, nahraje ho, spustí — máš URL
```

A hlavně *linka s bránou* (CI/CD), aby to nešlo nasadit ručně a pod tlakem — přesně ten omyl, který za pětačtyřicet minut zabil Knight Capital (kap. 21). Do repozitáře přidáš recept na linku:

```
# .github/workflows/deploy.yml (zkráceně)
on: { push: { branches: [main] } }
jobs:
  linka:
    steps:
      - run: pytest -q          # ◆ BRÁNA: neprošlo → STOP, nedeployuj
      - run: fly deploy         # jen když testy zelené
```

Ta jediná podmínka „`fly deploy` až *po* zeleném `pytest`“ je pakt z kapitoly osm zalitý do betonu: „nenasadím nic neověřeného“ přestane být tvoje předsevzetí, které v pátek večer poruší únava, a stane se zdí, kterou neobejdeš.

→ kap. 21: linka veze bednu (na rozdíl od roury z kap. 17, která skládá příkazy). `dev` → `stage` → `prod`*: nikdy nenasazuj rovnou naostro. K červenci 2026 Fly.io účtuje po vteřinách; minimální always-on stroj vyjde kolem 1,94 $ měsíčně, plný free tier pro nové účty zrušen. Alternativy (Render, Railway) fungují stejně — Dockerfile a* `deploy`.

TEENAGER · MECHANISMUS

Poslední zastávka: služba běží — teď musí o sobě *mluvit*, když spíš. Tři hlásné trouby z kapitoly jednadvacet:

```
LOGY     „co se právě stalo?"  — deník událostí,
čteš, když je zle.
METRIKY  „jak si vede?"  — her/s, latence, podíl
chyb → dashboard.
ALERTY   „vzbuď mě!"  — když podíl výher vypadne
z 64-69 %, přijde
         zpráva. Alert je budík, ne deník: obrací
směr — systém
         volá tebe, ne ty jeho.
```

Ten alert na invariant je půvabný: je to *tentýž* test z kroku 4, jen přesunutý z linky do běžícího provozu. Na lince hlídá, že se služba nasadit smí; v produkci hlídá, že pořád počítá to, co má. Kdyby drift (kap. 21) nebo cizí zásah odklonil číslo od dvou třetin, budík zazvoní dřív, než si toho někdo všimne. To je rozdíl mezi hračkou a produkcí, vyjádřený jedním pravidlem nad jednou metrikou.

→ kap. 21 niť klamu přeživších: monitoring měří jen to, co doletělo. *Dashboard ti hrdě ukáže hry, které se* odehrály *— ale mlčí o návštěvnících, kterým se stránka ani nenačetla. Měř i* absenci*, ne jen přítomnost; letadlo, které se nevrátilo, v grafu není.*

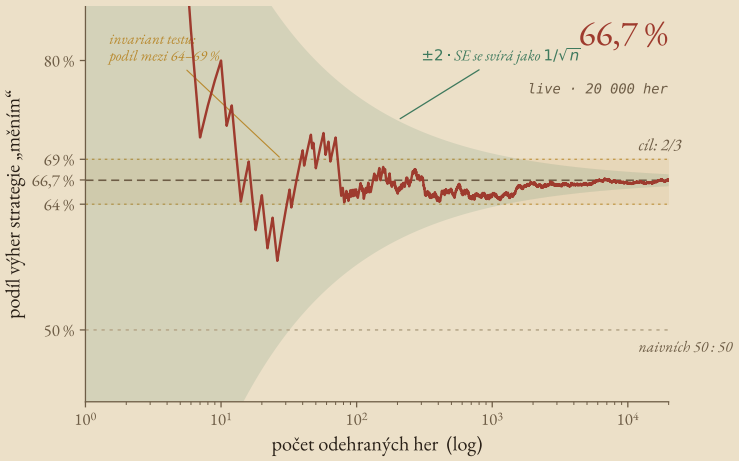
DEMO

`lesk-a-bida-vibecodingu.gaussalgo.com/d/capstone-monty`

Capstone naživo: otevři veřejnou službu tří dveří a sleduj, jak se podíl výher „měním“ usazuje na 66,7 %, zatímco se pásmo ±2·SE svírá jako $1/\sqrt{n}$. Běží dál, i když zavřeš oči — přesně o tom je celé dějství inženýr.

Einstein: proč se ten dashboard smí usadit

Erdős chtěl vědět *proč*. Batole a teenager postavili službu; Einstein teď doloží, že to číslo na dashboardu není zbožné přání, ale matematická jistota — a proč se dá *testem* hlídat interval 64–69 %.



Živý dashboard „na produkci“. Sanguine křivka je běžící podíl výher „měním“, jak přibývají hry návštěvníků (osa logaritmická). Zelené pásmo je teoretický interval $\pm 2 \cdot \mathrm{SE}$ kolem dvou třetin; svírá se jako $1/\sqrt{n}$. Okrové pásmo 64–69 % je invariant testu *z kroku 4: co linka hlídá, aby se do produkce nedostalo. Velké číslo vpravo je aktuální odhad; po dvaceti tisících hrách 66,7 %. Naivních 50 : 50 leží celou dobu mimo — tam, kde nikdy nebylo.*

EINSTEIN · MATEMATIKA — přeskočitelné

Proč se usadí — zákon velkých čísel. Každá odehraná hra je náhodný pokus: výhra strategie „měním“ (1) s pravděpodobností $p = 2/3$, prohra (0) jinak. Číslo na dashboardu je prostý průměr n takových nul a jedniček, $\overline{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$. Slabý zákon velkých čísel říká, že tento průměr konverguje podle pravděpodobnosti k pravé hodnotě:

$$\overline{X}_n \longrightarrow p = 2/3 \quad \text{pro} \quad n \to \infty.$$

To je matematická záruka, že dashboard nemůže dělat nic jiného, než se časem usadit na dvou třetinách — ať mu tisícovka doktorů věří, nebo ne.

EINSTEIN · MATEMATIKA — přeskočitelné

Jak rychle — a proč zrovna interval 64–69 %. Zákon velkých čísel říká *že* se usadí; centrální limitní věta říká *jak rychle*. Pro průměr n nezávislých pokusů s rozptylem $p(1-p)$ je směrodatná chyba odhadu

$$\mathrm{SE}(n) = \sqrt{\frac{p(1-p)}{n}} \propto \frac{1}{\sqrt{n}}.$$

Pro $p = 2/3$ vyjde po deseti tisících hrách $\mathrm{SE} \approx 0{,}47$ procentního bodu, takže pásmo $\pm 2 \cdot \mathrm{SE}$ je jen asi $\pm 0{,}94$ pb kolem 66,7 %. Odtud interval testu: 64–69 % je *bezpečně širší* než toto teoretické kolísání, takže poctivá služba jím po deseti tisících hrách projde skoro jistě — a špatná (třeba Monty Fall na 50 %) jím neprojde nikdy. Test tedy neměří tvůj kód proti dojmu, ale proti spočítanému intervalu: to je celý rozdíl mezi „vypadá to dobře“ a „ověřil jsem to“.

*$\pm 2 \cdot \mathrm{SE}$ je zhruba 95% interval spolehlivosti (přesnější 1,96). Volba mezí testu je inženýrské rozhodnutí: dost úzké, aby chytily vážnou chybu (50 %), dost široké, aby na správném kódu nezčervenaly náhodou (*flaky *test). Nastavit invariant na 66,6–66,8 % by byla past: poctivá služba by v něm sem tam propadla čirou náhodou. → kap. 14, 21: tentýž vzorec $1/\sqrt{n}$ jsme slíbili a tady je nasazený.*

EINSTEIN · MATEMATIKA — přeskočitelné

Kolik ten večer stál — bod zvratu, ne dojem. Poslední kámen je ekonomika, protože „levné“ má být číslo, ne pocit (kap. 30 ji rozvine). Agent tě stál tokeny: řekněme, že build spotřeboval T tokenů; při ceně Opusu 4.8 (5 $ / 25 $ za milion vstup / výstup, červenec 2026) je to pár dolarů. Server stojí zhruba $c \approx 1{,}94$ $ měsíčně. Celkový náklad prvního měsíce je tedy

$$N_1 = \text{tokeny} \cdot \text{cena za token} + c.$$

Pointa není číslo, ale *že ho umíš spočítat předem*. Kdo neví, kolik ho stojí jeden uživatel, toho první virál položí účtem — náklad patří na dashboard vedle latence. Rozhodnutí „agent, nebo napíšu sám“ a „cloud, nebo lokál“ jsou pak bod zvratu, ne víra: spočítej, neodhaduj.

→ kap. 30 (dějství vědec v praxi): ekonomika tokenů, kontextové okno jako rozpočet, kaskády levný↔drahý model, kdy se vyplatí lokál. Zde stačí reflex: každé zavolání modelu je peníze, každý běžící stroj je peníze — obojí měř, neodhaduj.

Co si odnést z capstonu

Poskládej to, protože tohle byl klenák celého dějství. Sedl sis k prázdnému adresáři a odešel s veřejnou službou — ne kouzlem, ale řemeslem, které má osm zastávek a u každé konkrétní příkaz. A přes všechnu tu novou rychlost jsi na každé zastávce ručil za výsledek: spec dřív než kód, testy jako zadání, commit před experimentem, brána před produkcí, alert na invariant. To je Willisonův *vibe engineering* v jednom večeru.

A tři věci si odnes do ruky. *Úsudek se nepřestěhoval, jen syntax:* agent napíše řádky, ale co stavět, jak to ověřit a kdy to nesmí ven, rozhoduješ ty — a přesně to je hodnota, která přežila pád ceny kódu na nulu. *Test je pakt, který únava neobejde:* invariant 64–69 % zabetonovaný do linky i do alertu hlídá pravdu za tebe, na lince i v provozu, ať spíš, nebo spěcháš. A *levné musí být číslo, ne dojem:* tokeny i server umíš spočítat předem, tak je počítej.

A capstone sám je celá kniha zhuštěná do jedné běžící věci. Tři dveře z kapitoly čtrnáct, jejichž matematiku popírala tisícovka doktorů; verzované jako v osmnáctce, otestované jako v devatenáctce, zocelené jako ve dvacítce, zabalené a nasazené jako v jednadvacítce — a postavené reálnými nástroji, které jsi osahal v kapitolách třiadvacet až šestadvacet. Na dashboardu se před očima usazuje 66,7 %, protože zákon velkých čísel jinak neumí. To, co kdysi stálo dopisovou válku pěti měsíců a odpoledne u počítače, dnes běží pro celý svět — i když u toho nikdo není. Tím se dějství inženýr uzavírá: uměl jsi poznat, že se pleteš (dějství vědec); teď z toho umíš postavit něco, co se nepletlo ani ve tři ráno, když jsi spal.

„Jak bych poznal, že se pletu?“

Konkrétní test k této kapitole je celé dějství v jedné otázce: *běží to, i když spíš?* Nasaď svůj capstone, zavři notebook a jdi na víkend. V pondělí se zeptej: poznám bez otevírání notebooku, jestli služba celou dobu fungovala, spadla, nebo tiše vracela nesmysly — a drží se pořád podíl výher u dvou třetin? Jestli musíš přijít, přihlásit se a projít logy ručně, abys to zjistil, máš odpověď: nepostavil jsi produkci, postavil jsi hračku, která měla přes víkend štěstí. Kapitola se plete jedině tehdy, když mi ukážeš tu službu bez jediného testu, bez brány na lince a bez alertu na invariantu — a ona přesto po dvou dnech bez dozoru přesně věděla, kdy se odchýlila od 66,7 %, sama tě vzbudila a ráno ti doložila, že celou noc počítala správně. Pak jsi buď postavil produkci, aniž bys věděl jak, nebo — a to je pravděpodobnější — ses ještě nepodíval dost pozorně na návštěvníky, které ti dashboard *nezapočítal*, protože se k tobě vůbec nedostali. Tak se podívej na tu díru, kterou graf mlčí; přesně tam bydlí hra, která se neodehrála.