XXI

Jak se z hračky stane produkce?

Po této kapitole poznáš rozdíl mezi programem, který běží, když se díváš, a systémem, který běží, když spíš. Naučíš se zabalit kód do kontejneru, poslat ho linkou, která ho bez prošlých testů nepustí dál, a nechat ho o sobě hlásit, když se pokazí. A vezmeš vše z tohoto dějství — verzování, testy, robustnost — a složíš z toho jeden nasazený projekt: tři dveře z kapitoly čtrnáct jako živou službu, na jejímž dashboardu se před očima usazuje 66,7 %.

> *Kontejner je jádrem vysoce automatizovaného systému pro přepravu zboží odkudkoli kamkoli s minimem nákladů a komplikací po cestě.*
>
> — Marc Levinson, The Box: How the Shipping Container Made the World Smaller and the World Economy Bigger, *Princeton University Press, 2006*

Krabice, která zmenšila svět

Šestadvacátého dubna 1956 vyplul z přístavu Newark v New Jersey přestavěný tanker jménem *Ideal-X* a zamířil do Houstonu. Na palubě vezl obvyklý náklad kapaliny v tancích — a k tomu osmapadesát velkých kovových beden, které tam ten den nikdo neuměl pojmenovat jinak než „návěsy bez podvozku“. Byly to první kontejnery, jak je dnes známe. Za jejich naloděním stál Malcom McLean, majitel kamionové firmy z Severní Karolíny — člověk, který o lodích nevěděl skoro nic a právě proto viděl to, co námořní dopravci přehlíželi celá desetiletí.

McLean nevynalezl bednu; bedny existovaly dávno. Vynalezl něco, co se nedá osahat: *rozhraní*. Do té doby se loď nakládala tak, že desítky mužů vláčely pytle, sudy a bedny jeden po druhém z mola do podpalubí a tam je ručně skládaly jako obří tetris — dny práce, hory rozkradeného a rozbitého zboží, loď zbytečně kotvící v přístavu místo na moři, kde vydělává. McLeanova myšlenka byla, že zboží se má naložit jednou, do bedny, a ta bedna už se nikdy neotevře, dokud nedojede na místo. Jeřáb ji zvedne z kamionu na

loď, z lodi na vlak, z vlaku na jiný kamion — a nikoho cestou nezajímá, co je uvnitř. Zajímají ho jen ty čtyři rohy.

Čísla, kterými se to změřilo, patří k nejpřesvědčivějším v dějinách logistiky. Naložit tunu volného zboží na loď stálo tehdy zhruba pět dolarů a třiaosmdesát centů. Naložit tunu v kontejneru na *Ideal-X* vyšlo podle McLeanových lidí na necelých šestnáct centů. To není o desetinu levněji, ani o polovinu — je to zhruba *sedmatřicetkrát* levněji. A tady je jádro, kvůli němuž ten příběh vyprávíme na začátku kapitoly o produkci: to kouzlo není v krabici. Krabice je hloupý plechový kvádr. Kouzlo je v tom, že všechny krabice mají *stejné rohy a stejné zámky* — takže jeřáb, podvozek i loď kdekoli na světě je umí chytit, aniž by věděly, kdo je vyrobil nebo co vezou. Hodnota nebydlí v předmětu; bydlí v tom, jak je předmět použit.

Prameny: M. Levinson, The Box *(2006), kap. 1 a 3;* Ideal-X *vyplul 26. 4. 1956 z Port Newark do Houstonu s 58 kontejnery. Náklad na tunu: 5,83 $ ručně vs. 15,8 centu kontejnerem (Levinsonova čísla) → poměr ≈ 37×.*

Poctivě: rohy a zámky ještě nebyly *celosvětově standardizované. ISO normy pro rozměry a rohové zámky přišly až v 60. letech;* Ideal-X *vezl McLeanovy vlastní 35stopé bedny, s pozdějším standardem nekompatibilní. Kouzlo rozhraní je* princip, *který McLean spustil — dotažen byl teprve dohodou na jediné normě o deset let později.*

Zastav se u toho posledního slova: *dohodou*. Krabice sama o sobě nezmenšila svět; svět zmenšila teprve chvíle, kdy se všichni shodli na jednom rozměru rohu. Do té doby měl každý dopravce vlastní bednu s vlastními zámky, a jeřáb z jednoho přístavu neuměl zvednout kontejner z druhého — pořád jsi měl palce proti centimetrům, jen v oceli. Teprve společné rozhraní udělalo z osamělého McLeanova nápadu globální síť. Přesně tuhle věc dělá i tvůj software, když z hračky přechází do produkce: přestane být osamělou bednou, kterou umíš zvednout jen ty na svém stole, a začne mít rohy, za které ho chytne stroj kdekoli a kdykoli, i když u toho nejsi. O těch rozích je celá tahle kapitola.

→ kap. 17: roura je totéž kouzlo o patro níž — příkazy se skládají, protože sdílejí jedno rozhraní (proud textu). Kontejner je kompozice pro celou infrastrukturu: standardní rohy místo standardního vstupu a výstupu.

Turbína, která se konečně vešla

V minulé kapitole jsem nechal otevřenou pátou jizvu. Je čas ji zavřít — protože McLeanovy rohy jsou přesně ten lék, který mi tehdy chyběl.

> Připomínám ránu z minula: turbína do Turecka, kterou jsme vyrobili přesně podle výkresu v centimetrech, zatímco zákazníkův základ byl v palcích. Přesná, drahá, a nevešla se — o rozdíl mezi palcem a centimetrem natažený přes celý stroj. Léta jsem si myslel, že to byl příběh o jednotkách. Když jsem ho vyprávěl počtvrté, došlo mi, že je to příběh o *rozhraní*. Nechybělo nám lepší měřidlo. Chybělo nám to, co McLean dal světu: jediná dohodnutá norma, do níž obě strany zapadnou, ať kreslí v čemkoli. Kdyby mezi naší konstrukcí a tureckým základem stálo standardní rozhraní — příruba s předepsanými rozměry, na kterou se turbína i základ musí trefit —, palec a centimetr by se potkaly na něm a nikdy uvnitř stroje. Jednotka té bolesti jsou pořád palce. Ale poučení už nejsou jednotky; je to *rozhraní*.

Všimni si, že Gimli, Mars i moje turbína byly děravé řetězy *bez společného rozhraní*: každý článek mluvil vlastním jazykem a mezera mezi jazyky propustila stroj. Kontejner tu mezeru zavírá tím, že donutí všechny články mluvit stejně. Přesně to udělá pro tvůj kód kontejner v příštím oddílu: „u mě to funguje“ je turbína v centimetrech; kontejner je ta příruba, na které se tvůj stroj a cizí server konečně potkají.

Hračka, nebo produkce?

Než se pustíme do rohů a zámků, potřebuješ jednu jasnou hranici — čáru, na které se pozná, jestli máš hračku, nebo produkci. Není to čára velikosti ani chytrosti. Vede jinudy, a jakmile ji jednou uvidíš, uvidíš ji všude.

Hračka běží, když se díváš. Produkce běží, když spíš.

To je celé. Program, který funguje, dokud sedíš u klávesnice, spouštíš ho ručně a hned vidíš, když spadne — to je hračka, i kdyby to byl geniální model za milion řádků. Systém, který běží dál, i když zavřeš notebook a jdeš na víkend, který se sám zvedne po pádu, sám tě vzbudí, když se pokazí něco, co se pokazit nemělo, a sám ti ráno poví, že celou noc počítal správně — to je produkce, i kdyby to bylo dvacet řádků. Rozdíl není v tom, co program *umí*. Rozdíl je v tom, co se stane, když u toho *nejsi*.

Tahle věta je v knize podruhé: poprvé zazněla v kap. 14 jako slib, že simulaci tří dveří jednou proměníme v produkci. Teď ten slib splníme.

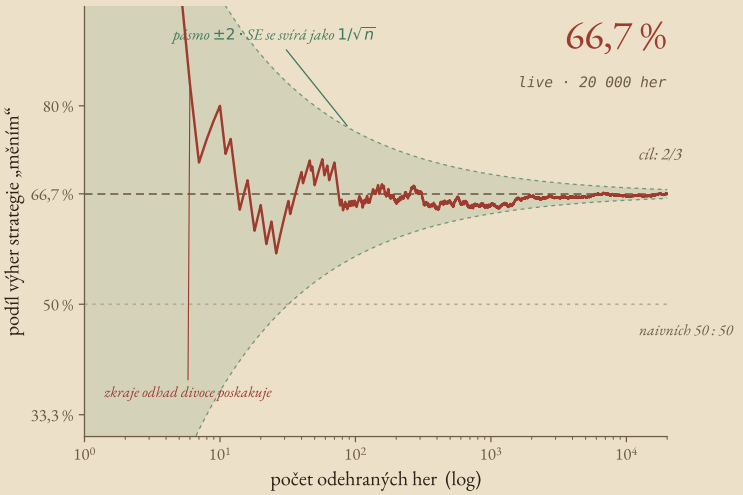
> *Vyzkoušej si to hned, než začneš cokoli stavět. Vypni si obrazovku na víkend a v pondělí se zeptej sám sebe: poznám vůbec, jestli to celou dobu fungovalo, spadlo, nebo tiše vracelo nesmysly? Jestli odpověď zní „netuším, musel bych se podívat a projít to ručně“ — nemáš produkci. Máš hračku, která měla zrovnou štěstí.*

Capstone: tři dveře jako veřejná služba

A teď to nejlepší, kvůli čemu tahle kapitola stojí na konci celého dějství. Vezmeme tu hračku, kterou jsi postavil v kapitole čtrnáct — pár desítek řádků, co mnohotisíckrát rozehrají hru o tři dveře a sečtou výhry —, a proměníme ji v produkci. Ne v myšlenkách; doopravdy. Nasadíme ji jako *veřejnou službu*, ke které se připojí kdokoli, odehraje svých pár her, a jeho výsledky se přičtou k dashboardu, na kterém se před očima všech usazuje podíl výher strategie „měním“ na dvou třetinách.

Zamysli se na chvíli, co to je za věc. Je to živý, běžící *důkaz* té samé matematiky, kterou v kapitole čtrnáct popírala tisícovka doktorů a s nimi Erdős. Nemusíš nikomu nic dokazovat dopisem ani rozhodovacím stromem; ukážeš na obrazovku, kde se každou vteřinou přičítají hry cizích lidí — a číslo se tvrdošíjně drží u 66,7 %, ať si o tom kdo myslí co chce. Dashboard je argument, který nikdy nespí a nikdy se nezalekne

titulu na druhé straně. Monitoring se tu stává tím, čím v kapitole čtrnáct byla simulace: strojem, který spor o pravdu rozhodne za tebe.



Hrdinský graf kapitoly — capstone dashboard „naživo“. 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 $1/\sqrt{n}$ — přesně ta zužující se soutěska z kap. 14, teď nakreslená na živých datech. Velké číslo vpravo nahoře je aktuální *odhad: po dvaceti tisících hrách 66,7 %. Naivních 50 : 50 leží celou dobu mimo pásmo — tam, kde nikdy nebylo.*

Tenhle projekt je *capstone* celé knihy: klenák, který spojuje oblouk. Vezme totiž vše, co ses naučil v dějství o inženýrství, a složí to do jednoho nasazeného kusu. *Verzování* z kapitoly osmnáct: kód žije v repozitáři, každá změna je commit, každý experiment větev. *Testy* z kapitoly devatenáct: napíšeš invariant „podíl výher „měním“ leží po deseti tisících hrách mezi 64 a 69 procenty“ a linka ho ohlídá za tebe. *Robustnost* z kapitoly dvacet: když návštěvník pošle nesmysl nebo spadne databáze, služba se nezhroutí, degraduje se s grácií. A rozhraní z tohoto podobenství: celé to zabalíš do kontejneru, který poběží na tvém stroji stejně jako na cizím serveru. Postav ⟶ otestuj ⟶ verzuj ⟶ nasaď ⟶ měř. Pět sloves, jedna běžící věc.

DEMO

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

Capstone naživo: otevři veřejnou službu tří dveří, odehraj svých pár her a sleduj, jak se *tvoje* hry přičtou k dashboardu, na kterém se podíl výher „měním“ usazuje na 66,7 %. Tentýž rozhodčí, který smířil Erdőse — teď běžící pro celý svět, i když zrovna nikdo nekouká.

A tady je ta dobrá zpráva, kvůli které za celé dějství stálo: nasadit takovou službu dneska nestojí ani přístav, ani jeřáb, ani McLeanovu odvahu. Stojí to večer, kontejner a pár řádků linky — nástroje, které v části II osaháš rukama. Poprvé v dějinách má jednotlivec logistiku, o jaké se námořním dopravcům roku 1956 nesnilo, a dostane ji zadarmo. Píseň je skutečně krásná: to, co kdysi zmenšilo svět, dnes leží ve svobodné podobě na tvém stole.

Řemeslo: kontejner, linka a hlásné trouby

Mezi hračkou a produkcí leží tři konkrétní řemesla: *zabalit* (kontejner), *dopravit bez úrazu* (linka) a *slyšet, když se něco děje* (monitoring). Vezmeme je po řadě.

Kontejner řádek po řádku

Kontejner v softwaru je McLeanova bedna přeložená do počítače: krabice, do níž zabalíš svůj program *se vším*, co potřebuje k běhu — jazyk, knihovny, nastavení —, zavřeš ji a ona pak běží stejně na tvém notebooku, na cizím serveru i v cloudu. „U mě to funguje“ přestane být výmluva, protože „u mě“ a „u tebe“ je odteď stejná bedna. Recept na tu bednu se píše do souboru, který si tady rozebereme řádek po řádku — v pseudopodobě, ať vidíš principy, ne syntaxi konkrétního nástroje.

TEENAGER · MECHANISMUS

Recept na kontejner (pseudo-Dockerfile) — čtyři věty, čtyři rohy bedny:

```
ZÁKLAD  python:3.12          # od čeho začínám: hotový podklad
                             #   s jazykem — nestavím od nuly

ZKOPÍRUJ  ./tri_dvere  /app  # co dávám dovnitř: můj kód
                             #   se přesune do bedny

PŘIPRAV  nainstaluj závislosti   # co se musí stát JEDNOU při
                                 #   balení: knihovny do bedny

SPUSŤ  server tri_dvere      # co se stane POKAŽDÉ při startu:
                             #   tímhle příkazem se bedna „zapne"
```

Rozdíl mezi PŘIPRAV a SPUSŤ je celé kouzlo. *Připrav* se odehraje jednou, když bednu balíš — nainstaluje se, co je potřeba, a zapečetí se dovnitř. *Spusť* se odehraje pokaždé, když bednu někdo otevře a zapne. Proto se kontejner chová všude stejně: to, co by se jinak lišilo stroj od stroje (verze knihovny, chybějící balík), je zavřené uvnitř a nikdo cestou to už nemění. Přesně McLeanova bedna: naložíš jednou, otevřeš na místě, a mezitím se nikdo nešťoural v obsahu.

V praxi: soubor se jmenuje `Dockerfile` *a klíčová slova jsou* FROM, COPY, RUN, CMD. *Nástroje Docker / Podman z něj postaví* image *(zapečetěnou bednu) a spustí* container *(běžící kus). Princip „zabal se vším a zapečeť" je ale starší i širší než Docker — jde o izolaci a přenositelnost, ne o konkrétní logo.*

Pozor na past: bedna má být co nejmenší a bez tajemství uvnitř (hesla ne do image!) — jinak vezeš přes půl světa kontejner plný zbytečností a klíčů od domu.

Linka, ne roura

Teď musíš bednu dopravit z tvého stolu do produkce — a to je práce pro *linku*. Dej pozor na slovo: v kapitole sedmnáct jsme měli *rouru*, kde se příkazy skládají za sebe a proud teče z jednoho do druhého. Linka je něco jiného: výrobní pás, po kterém *jede artefakt* — tvoje bedna — a na každém stanovišti se s ní něco stane. Roura skládá příkazy; linka veze bednu. A na jednom stanovišti té linky stojí *brána*.

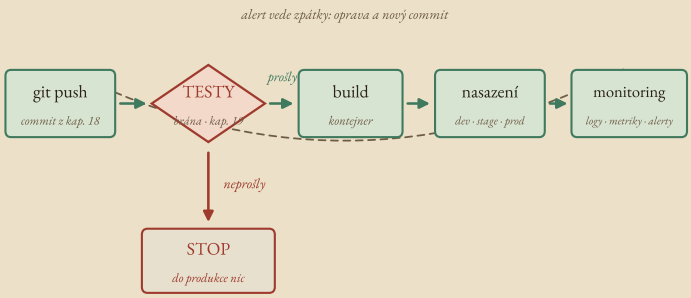
TEENAGER · MECHANISMUS

Produkční linka (CI/CD) — pět stanovišť a jedna brána:

```
1. PUSH      pošleš commit (kap. 18) — linka se
rozjede sama

2. TESTY  ◆  BRÁNA: pustí testy z kap. 19.
Prošly? jeď dál.
             Neprošly? → STOP. Do produkce se
nedostane NIC.

3. BUILD     z kódu se postaví kontejner (bedna z
receptu výše)

4. NASAĎ     bedna jede do prostředí: dev →
stage → prod

5. MONITOR   služba běží a hlásí o sobě: logy,
metriky, alerty
```

Slovo *brána* na druhém řádku je celý smysl linky. Bez ní bys nasazoval ručně a pod tlakem — a přesně tam vznikají katastrofy (viz glosa). S ní se do produkce nedostane žádná změna, která neprošla testy, a to *bez ohledu* na to, jak moc spěcháš nebo jak jsi si jistý. Linka je pakt z kapitoly osm zabetonovaný do stroje: „nenasadím nic neověřeného“ už není tvoje předsevzetí, které v pátek večer poruší únava — je to zeď, kterou neobejdeš.

LINKA (ne roura): od commitu do produkce jedním směrem



Linka od commitu do produkce: `push` → brána testů → `build` → `nasazení` → `monitoring`. *Zelené šipky = prošlo. Když testy neprojdou (sanguine větev), linka se zastaví u brány a do produkce nejde nic. Přerušovaná smyčka nahoře: alert z monitoringu vede zpátky k novému commitu — zpětná vazba z kap. 19 uzavřená do kruhu.*

Krátká zastávka u třetího řádku, prostředí `dev` → `stage` → `prod`. Nikdy nenasazuj rovnou do ostrého provozu; provoz je jeviště, ne zkušebna. *Dev* je tvůj ponk, kde se smí rozbíjet. *Stage* je generálka: kopie produkce co nejvěrnější, kde bednu vyzkoušíš „naostro nanečisto“, dřív než se jí dotkne zákazník. *Prod* je premiéra. Táž bedna projde všemi třemi beze změny — to je zase to McLeanovo rozhraní: co se osvědčilo

CI/CD — průběžná integrace a průběžné nasazování: linka, která od commitu do produkce nepustí nic, co neprošlo testy. V praxi: GitHub Actions, GitLab CI, Jenkins.

na generálce, poběží v premiéře stejně, protože je to *doslova ten samý kontejner*. A že se premiéra nesmí hrát bez generálky ani bez brány, o tom mluví nejdražší postmortem v dějinách softwaru — vedle v gloze.

Knight Capital, 1. 8. 2012. Obchodní firma nasadila nový kód na osm serverů — ale na jeden z nich se nedostal. Ten osmý oživil starý, dávno zapomenutý kus programu, který se měl nikdy nespustit. Za pětačtyřicet minut *vyslal automat čtyři miliony chybných příkazů, nakoupil pozice za miliardy a firmu připravil o zhruba* 440 milionů dolarů *— a tím ji prakticky zabil. Chyběla brána a chybělo pravidlo „táž bedna všude“: nasazení bez pojistky proměnilo jeden nedbalý deploy v konec společnosti. Přesně proti tomuhle stojí linka.*

Hlásné trouby: logy, metriky, alerty

Nasadit nestačí. Vzpomeň si na hranici: produkce běží, když spíš — a když spíš, potřebuješ, aby ti systém *sám* řekl, co dělá. K tomu slouží tři různé hlásné trouby a lidé je pletou, takže si je rozliš hned.

TEENAGER · MECHANISMUS

Tři způsoby, jak systém mluví — každý na jinou otázku:

```
LOGY     „co se právě stalo?"  — deník událostí,
řádek po řádku.
         Čteš je, když už víš, že je zle, a
hledáš PROČ.

METRIKY  „jak si vede?"  — čísla v čase: kolik
her za minutu,
         jaká latence, jaký podíl chyb. Z nich je
dashboard.

ALERTY   „vzbuď mě!"  — pravidlo nad metrikou:
když podíl chyb
         přeleze práh, přijde zpráva. Alert je
budík, ne deník.
```

Klíč je poslední řádek. Log, který nikdo nečte, je němý; metrika, na kterou se nikdo nedívá, je slepá. Teprve *alert* mění pasivní záznam v aktivní hlídku — obrací směr: ne ty se ptáš systému, ale systém volá tebe. Bez alertu je „produkce běží, když spíš“ jen zbožné přání: budeš spát dál i ve chvíli, kdy hoří.

→ kap. 7: alert a limit jsou protilavinové zábrany. Reply-all bouře i přetížená služba jsou tentýž zesilovač, který nezná směr — alert je hlídka, která ho zastaví dřív, než lavina nabere rychlost.

Pozor na únavu z alertů*: budík, který zvoní pořád, se přestane poslouchat. Alert jen na to, co vážně bolí — jinak si vypěstuješ hluchotu vůči vlastnímu systému.*

A teď niť, kterou si z tohoto dějství musíš odnést až do praxe, protože je zákeřná: *monitoring měří jen to, co doletělo.* To je klam přeživších z kapitoly třináct, převlečený do telemetrie. Tvůj dashboard ti hrdě ukáže latenci požadavků, které *dorazily* — ale mlčí o těch, které se ke tvému serveru vůbec nedostaly, protože spadla síť po cestě. Ukáže ti chybovost her, které se *odehrály* — ale nezapočítá návštěvníky, kterým se stránka ani nenačetla, a tak žádnou hru nezaznamenali. Stejně jako Wald nepočítal letadla, která se nevrátila, ty nepočítáš požadavky, které nedoletěly. A protože dashboard vypadá tak konkrétně a uklidňujícně, uvěříš mu — a klam přeživších tě dostane přesně tam, kde se cítíš nejjistěji.

→ kap. 13: Waldova nevrácená letadla. Monitoring vidí jen „vrácené stroje“ — požadavky, které doletěly a nechaly stopu. Díra, kterou nevidíš, je nejnebezpečnější: neexistuje v grafu, tak neexistuje ve tvé hlavě. Lék: měř i absenci *(kolik čekaných požadavků nepřišlo), ne jen přítomnost.*

Když v systému bydlí model

Poslední řemeslo dílny patří tvé době. Jestli je součástí tvé produkce jazykový model — a to bude čím dál častěji —, přibývají ti čtyři starosti,

které klasický software nezná. Vypíšu ti je natvrdo, protože se na ně zapomíná až do prvního účtu nebo prvního výpadku.

TEENAGER · MECHANISMUS

Čtyři věci navíc, když v produkci běží model:

```
DRIFT      Svět se mění, model zůstal. Vstupy, na
které kdysi
           odpovídal dobře, dnes vypadají jinak
→ tichý úpadek
           kvality. Měř kvalitu i po nasazení, ne
jen před ním.

NÁKLADY    Každé zavolání modelu stojí tokeny =
peníze. Metrika
           „kolik nás stojí jeden uživatel" patří
na dashboard
           vedle latence — jinak tě první virál
položí účtem.

LATENCE    Model odpovídá pomalu a nerovnoměrně.
Nastav limit
           a rozhodni, co dělat, když ho překročí
(viz fallback).

FALLBACK   Model je vzdálená služba, která může
být pryč. Co
           uděláš, když mlčí? Máš záložní,
jednodušší odpověď —
           nebo spadneš celý? Graceful
degradation z kap. 20.
```

Všimni si, že tři ze čtyř jsou staré známé z minulé kapitoly — latence, fallback a náklad jsou jen konkrétní tvary robustnosti. Nové je jen slovo *drift*: model se nezhoršuje hlučně jako spadlý server, ale tiše, jako by ztrácel formu. A tichý úpadek je přesně to, co produkce, která „běží, když spíš“, musí umět zachytit sama.

drift — model stárne vůči měnícímu se světu, i když se v něm nezměnil jediný parametr. Souvisí se zhroucením modelu z kap. 5 i s Goodhartem z kap. 16: metrika, která kdysi měřila kvalitu, ji po čase měřit přestane. Detaily ekonomiky tokenů a fallbacků žijí v části II.

Věž: zákon velkých čísel, SLO a fronty

EINSTEIN · MATEMATIKA — přeskočitelné

Proč se dashboard usazuje — zákon velkých čísel. Každá odehraná hra je náhodný pokus: výhra (1) s pravděpodobností $p = 2/3$, prohra (0) jinak. Odhad, který dashboard ukazuje, 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 — centrální limitní věta a pásmo $1/\sqrt{n}$. 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

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

Tohle je ta zužující se soutěska na dashboardu, teď s číslem. Pro $p = 2/3$ vyjde po sto hrách $\mathrm{SE} \approx 4{,}7$ procentního bodu, ale po deseti tisících už jen $0{,}47$ — desetkrát užší pásmo za *stonásobek* her. Proto se na živém dashboardu vyplatí hlídat, kolik dat už máš: dokud je pásmo $\pm 2 \cdot \mathrm{SE}$ široké, číslu se věřit nedá; teprve když se soutěska sevře, „usadilo se“ přestává být dojem a stává se tvrzením.

$\pm 2 \cdot \mathrm{SE}$ je zhruba 95% interval spolehlivosti (přesněji 1,96). Zelené pásmo na hrdinském grafu je přesně tohle: kam s 95% jistotou padne odhad z n her, když je pravda $2/3$. *→ kap. 14: tentýž vzorec jsme tam slíbili nakreslit na dashboard — tady je splněno. → kap. 22: zákon velkých čísel a CLT tam dostanou plné jméno u kalibrace.*

EINSTEIN · MATEMATIKA — přeskočitelné

SLO a rozpočet chyb. Produkce potřebuje *cíl*, jinak „běží dobře“ nic neznamená. **SLO** (service level objective) je takový cíl vyslovený číslem: například „99,9 % požadavků vyřídíme do 200 ms“. Doplněk do stovky je *rozpočet chyb* (error budget): když je cíl 99,9 %, smíš selhat v 0,1 % případů — a to je rozpočet, který můžeš *utratit*.

$$\text{rozpočet chyb} = 1 - \text{SLO} = 0{,}1\% \text{ za období.}$$

Pointa je v tom slově *rozpočet*: chyba přestává být hanba a stává se měnou. Dokud rozpočet nevyčerpáš, smíš riskovat, nasazovat, experimentovat. Když ho vyčerpáš, zmrazíš nasazování a soustředíš se na stabilitu. Je to Goodhart z kapitoly šestnáct obrácený ve prospěch: místo abys honil nedosažitelnou nulu chyb (a metrika se zvrhla), stanovíš *přijatelnou* míru selhání a řídíš se jí jako penězi.

Rozdíl SLO vs. SLA: SLO *je tvůj vnitřní cíl,* SLA *(agreement) je smluvní slib zákazníkovi s pokutou za porušení. SLO se drží pod SLA, aby ti zbývala rezerva. Nulové SLO neexistuje: 100 % spolehlivost je Damoklés z kap. 20 — nekonečně drahá a stejně nedosažitelná.*

EINSTEIN · MATEMATIKA — přeskočitelné

Littleův zákon: proč fronta roste. Poslední kámen do věže je pravidlo o frontách, které tě zachrání před nejčastější záhadou produkce („proč to najednou tak zpomalilo?“). **Littleův zákon** svazuje tři veličiny jednou rovnicí, která platí za velmi mírných předpokladů úplně vždycky:

$$L = \lambda \cdot W.$$

L je průměrný počet požadavků v systému (délka fronty i s tím, co se právě zpracovává), λ je příchozí tok (požadavků za vteřinu) a W je průměrná doba, kterou v systému každý stráví. Čti to takhle: když ti *zdvojnásobí* nápor λ a ty neumíš zrychlit obsluhu W, fronta L se ti zdvojnásobí — a s ní čekání. A jakmile obsluha nestíhá, W začne růst samo od sebe, protože se čeká ve frontě na čekající — a máš lavinu z kapitoly sedm, jen v jiném kabátě. Littleův zákon je důvod, proč se latence nekazí plynule, ale zlomí naráz.

Little (John D. C. Little, 1961): $L = \lambda W$ *platí pro stabilní systém nezávisle na rozdělení příchodů i obsluhy — proto je tak mocný. Praktický důsledek: chceš-li kratší čekání* W *při daném toku* λ*, musíš zkrátit frontu* L *— víc obsluhy, nebo méně práce na požadavek.*

Co si odnést z produkce

Poskládej to dohromady, protože tahle kapitola byla klenák. McLean nezmenšil svět lepší krabicí; zmenšil ho *rozhraním* — společnými rohy, za které svět chytne cokoli, aniž by věděl, co je uvnitř. Tvůj software přechází z hračky do produkce přesně v té chvíli, kdy dostane takové rohy: kontejner, do kterého se zabalí se vším a poběží všude stejně; linku s bránou, která ho bez prošlých testů nepustí dál; a hlásné trouby, kterými o sobě dá vědět, i když spíš.

A tři věci si odnes do ruky. *Zabal a zapečeť:* kontejner ukončí „u mě to funguje“, protože „u mě“ a „u tebe“ je odteď táž bedna — a táž bedna

projde generálkou i premiérou beze změny. *Postav bránu:* nasazení bez testovací brány je Knight Capital, který za pětačtyřicet minut zabil firmu; linka je tvůj pakt z kapitoly osm zalitý do betonu, který únava neobejde. A *měř tak, abys viděl i to, co nedoletělo:* dashboard je mocný, ale je to klam přeživších v přímém přenosu — počítá jen vrácená letadla, tak si hlídej i tu díru, kterou graf mlčí.

A capstone, který jsme právě principiálně provedli, je celá kniha zhuštěná do jedné běžící věci: tři dveře z kapitoly čtrnáct, verzované jako v osmnáctce, otestované jako v devatenáctce, zocelené jako ve dvacítce, zabalené do kontejneru a nasazené za linkou — a na dashboardu se před očima usazuje 66,7 %, protože zákon velkých čísel jinak neumí. To, co tisícovka doktorů popírala dopisy, teď běží pro celý svět, i když u toho nikdo není. Přesně to je rozdíl mezi hračkou a produkcí — a přesně to je odměna za celé dějství inženýr. V části II tuhle službu postavíš vlastníma rukama, skutečnými nástroji, krok za krokem.

„Jak bych poznal, že se pletu?“

Konkrétní test k této kapitole je nejjednodušší v celé knize a nejtěžší ho přežít: tvoje appka běží — *vypni si obrazovku na víkend*. V pondělí se zeptej: poznám, jestli celou dobu fungovala, spadla, nebo tiše vracela nesmysly? Jestli musíš otevřít notebook a projít to ručně, abys to zjistil, máš odpověď: nemáš produkci, máš hračku, která zatím měla štěstí. Kapitola se plete jedině tehdy, když mi ukážeš systém bez brány, bez alertu a bez kontejneru, který jsi nechal dva dny běžet bez dozoru — a on přesně věděl, kdy se pokazil, sám tě vzbudil a ráno ti doložil, že celou noc počítal správně. Pak jsi buď postavil produkci, aniž bys věděl jak, nebo — a to je pravděpodobnější — ses ještě nedíval dost pozorně na to, co ti dashboard *neukazuje*. Tak se podívej na tu díru, kterou mlčí; přesně tam bydlí letadlo, které se nevrátilo.