SAMPLE

Building the Software Infrastructure of Tomorrow

The Invisible Middle Where Every Useful Lie Is Made

Aiman JoeHanz

Free sample: Introduction & Chapter 1

putrabooks.github.io

Buy the full book:

Carousell carousell.sg/p/building-the-software-infrastructure-of-tomorrow-ebook-pdf-1467073435

Ko-fi

ElevenReader

Payhip

Contents

Introduction

Chapter 1: The Kernel — The Machine's Resident Government

Chapter 2: The Process — A Manufactured Life

Chapter 3: Memory — The Borrowed Instant

Chapter 4: Concurrency — Many Hands at One Table

Chapter 5: The File — Persistence and Its Promises

Chapter 6: The Wire — The Layered Conversation

Chapter 7: The Database — State That Survives Everyone

Chapter 8: The Machine That Makes Machines — Virtualization and the Container

Chapter 9: The Distributed System — One Program, Many Clocks

Chapter 10: The Long Run — Rot, Portability, and What Survives

A Last Word

About this Series

Connect

This free sample contains the Introduction and Chapter 1. The complete book contains the full contents above.

Introduction

Beneath every application the reader has ever opened, and above every chip the reader will never see, there is a middle country, and almost nobody visits it. Software infrastructure is that country: the operating systems, the memory managers, the file systems, the network stacks, the databases, the virtual machines, and the distributed runtimes that stand between the program someone wrote and the electricity someone else engineered. This volume descends into that middle country and stays there, because the infrastructure has an anatomy like any other engineering layer, and the reader who commands it, or depends on it, or debugs it at an hour no one wants to name, works blind without that anatomy.

Two earlier volumes of this series toured the neighboring floors. One walked the whole product craft from altitude, and one descended into the making of software, the engineering layer where chosen intent becomes running code. This book goes below the application entirely, to the system software that makes application programs possible at all, and it makes a claim the tour will keep proving: the infrastructure is not plumbing, it is a set of arguments, every one of its abstractions an argument about what should be hidden and what should be paid for, and the arguments are still being won and lost.

The tour runs in the order a machine's day runs. There is the kernel, the machine's resident government, the referee that owns the hardware and lends it out. There is the process, the manufactured life, a running program with a name, a budget, and a schedule. There is memory, the borrowed instant, the shared and persistent lie that every program lives alone in a vast, tidy address space. There is concurrency, many hands at one table, the craft of doing several things at once without the table breaking. There is the file, persistence's oldest promise, and the honest account of how fragile that promise really is.

There is the wire, the layered conversation by which distant machines pretend to share a wire. There is the database, the layer built when files proved not enough, the craft of keeping shared state alive through crashes and contention. There is the machine that makes machines, virtualization and the container, the abstraction that ate the machine itself. There is the distributed system, one program stretched across many clocks, where the wire becomes an internal organ and every assumption inherits a failure mode. Closing the tour comes the long run, the layer's strange relationship with time, in which code never wears out but rots just the same, and the craft's judgment finally concerns what survives.

The series' working rules hold on this floor as on every floor: no brand flattered, no invented number, no trend chased, and plain vocabulary throughout, because the folklore of this layer is thick in its own way, thick with certification mythologies and thick with the belief that the infrastructure is a static fact rather than an ongoing argument. The reader will find no secret tuning tricks in these pages, and no incantations, because the infrastructure has none, only abstractions with prices, and disciplines for reading those prices honestly: the discipline of knowing what an abstraction hides, of measuring before optimizing, of suspecting one's own wiring before suspecting the platform, of treating every shared resource as a queue waiting to be found. Each discipline is learnable and unglamorous, which is why folklore keeps outsourcing them to ceremonies and vocabularies, and the chapters ahead treat them as working tools instead, one per organ.

One framing deserves the opening's last word, because the whole tour depends on it: infrastructure is the craft of useful lies. A memory address is a lie told to a program about where its data lives. A file is a lie about permanence, told across a spinning or flashing medium that guarantees neither. A distributed system is a lie about there being one system at all. The lies are not vices; they are load-bearing, each one hiding a cost the program below could not bear to negotiate, and buying the program above a freedom it could not otherwise afford.

But every useful lie has a failure surface exactly where the truth was hidden, and the working professional's whole competence, on this layer, is knowing where each lie bends: which promises the abstraction quietly breaks, which guarantees arrive only in pairs that cannot both be collected, and which corners of the machine still speak the raw truth the lies were built to suppress. The chapters ahead are, chapter by chapter, the catalog of each lie and its particular bending point, and the reader who finishes them will never again be surprised by where the infrastructure bleeds.

The infrastructure has spawned a large literature of its own, and the reader deserves a map of where this volume sits within it. The layer's books tend to run to three kinds: the manual, which documents one product's knobs and expires with its version; the textbook, which proves the theorems beautifully and leaves the working reality at the door; and the war story, which is honest about one career but transfers only where careers rhyme. This series has chosen the fourth path: the anatomy, written about no product, citing no benchmarks, naming no vendors, describing only the working pattern that survives beneath every version, every theorem, and every career, and the price of that choice is that every claim in these pages must stand on its own reasoning, checkable against the reader's own machines and working life, the

one laboratory guaranteed shared.

The invitation is constant throughout: read each chapter against a system the reader actually runs, find where the described lie bends and where it holds, and trust the bending point rather than any name, because the craft lives in the bending points, and always did.

A note on the reading order belongs here, because the chapters were built to stand alone but were also built in a line, and the line has a logic worth one sentence of respect: the tour begins where the machine begins, at the government that boots first, and follows the work upward through the borrowed, the shared, and the persisted, until the last chapters stand at fleet scale looking at decades, so the reader short on time may read the first and last chapters as a spine, the government and the long run, and the reader with an afternoon is invited to read straight through, since every chapter hands its tools to the next, and the middle chapters, memory, concurrency, the file, the wire, the database, contain the tools the last chapters quietly use.

However the tour is taken, one promise governs the whole route, and it is made here in the opening so the chapters never need to repeat it: nothing in this book is secret, nothing is patented by its vocabulary, and nothing requires a prior apprenticeship beyond the willingness to ask, of every shining machine, the one plain question this series exists to teach, which is what is really happening underneath, and the chapters ahead are simply that question, asked ten times, of the ten load-bearing lies that carry everything else, and answered, each time, with the working truth of where they bend.

Chapter 1: The Kernel — The Machine's Resident Government

Every machine needs a government, and the kernel is the one it gets: the first program loaded at startup, the permanent resident, the sole owner of the hardware, and the referee every other program must petition for access to anything real. The kernel's existence answers a brutal engineering problem. A machine has one processor, or a handful, one physical memory, one disk, one wire, and many programs that each believe they own all of it; without an arbiter, the first two programs to touch the same memory would corrupt each other into silence. So the machine's designers built a border, and a bureaucracy for crossing it: the processor itself runs in two moods, a privileged mode in which everything is permitted and a restricted mode in which most things are forbidden, and the kernel lives on the privileged side of that border while every application lives on the restricted side.

The working meaning of the infrastructure begins right there, at that border crossing, and this chapter walks it slowly, because every layer above is a consequence of the kernel's founding decision.

The border crossing has a grammar, and it is called the system call. When a program wants a file opened, a byte of memory claimed, a message sent, a clock read, it does not reach for the hardware; it executes a special instruction that hands control to the kernel, states its request, and waits in line like every other petitioner. The crossing has a price, and the working professional should know its shape: switching between moods flushes work the processor had in flight, so a system call costs more than an ordinary instruction by orders of magnitude, and the programs that suffer are the ones that cross the border in a frenzy, asking in a thousand small petitions for what one larger request could have carried.

The optimization instincts of this layer, batching, buffering, and keeping work on the application's side of the line, are all ways of paying the border toll in bulk, and they recur in every chapter that follows, because every organ of the infrastructure charges for crossing it.

The kernel's first and deepest lie is the machine itself: it tells every program that the machine is its machine alone. Each process is handed a private continent of memory, protected by hardware the kernel programs, so that one program's wild pointer cannot touch another's data; the processor is sliced into time, handed out in turns so short that each program feels the whole machine idling under it; and the disk's single surface is carved into name-spaced territories, so that no program sees another's files unless both agree. Isolation,

multiplexing, and protection are the three founding services, delivered as a lie that every working program needs, and the craft's honesty lives in remembering that the private continent is an illusion sustained by constant bookkeeping, which is why a machine with too many programs does not run each slower in a polite linear way but begins to thrash, spending the majority of its turn doing the bookkeeping of the illusion rather than the work the programs came for.

The chapter closes on the kernel's quiet power, because the working professional meets that power mostly as absence. The machine that stays up for months, that survives a failing disk, that hands the midnight failure to a journal and comes back whole, is the kernel's doing, and so is the machine that stalls on the morning after a configuration change, because the government that owns everything also fails in charge of everything. The working disciplines are three: give the kernel honest resources, since starvation thrashing and cron-jammed queues are usually resource problems wearing kernel clothing; distrust any program that bypasses the border crossing, since drivers and privileged helpers run on the government's side and can burn the whole city; and when diagnosing a stall, look first at what the machine's referees are refusing, the queue lengths, the memory pressure, the wait channels, before blaming any single program's code.

The government is invisible in proportion to how well it governs, and the reader who internalizes its ledger will read every later chapter, and every real outage, with a cleaner eye.

Two organs of the government's day deserve their own paragraphs, because they explain behaviors the working professional otherwise meets only as mystery. The first is the interrupt, the machine's tap on the shoulder: hardware cannot wait politely for the kernel's attention, so the devices themselves are granted the power to seize the processor mid-instruction, freezing whatever program was running while the kernel notes the event, queues the work it implies, and returns the processor to its interrupted tenant. The interrupt is why a program's execution time wobbles under load that is not visibly its own, and it is why the hard real-time question, can this code be guaranteed to respond within a bound, is answerable only by machines engineered to bound their own interruptions; the ordinary machine, generous with its devices, cannot promise it, and any vendor claiming otherwise for commodity hardware is selling a rounding of statistics.

The second organ is boot itself, the quiet coup by which the government takes office: a small built-in program wakes the processor at a fixed address, inventories the machine, loads the kernel from a designated place, hands it

the inventory, and retires, and the entire ceremony runs in seconds that the industry has spent decades making boring, since a boot that needs attention is a machine that cannot recover itself, and the whole point of infrastructure, visible here in its purest form, is to need no one at two in the morning.

The border's traffic deserves one more honesty, because the working day sees its residue constantly and rarely names it. Every crossing the kernel handles costs the machine a small bookkeeping, and the modern kernel keeps its bookkeeping cheap by doing, per crossing, as little as possible: the request is validated, the work is done or deferred, and the reply is recorded, with anything heavier postponed to a soft-interrupt moment when the machine has spare attention. The professional reading a performance report should therefore learn to see crossing counts where the folklore sees magic: the server that handles each arriving message with a separate petition, reading, writing, yielding, and sleeping in turn, is paying the toll a dozen times where a batched design would pay once, and the remedy, everywhere from database access to log writing, is the same architecture of accumulation, doing many things per crossing rather than one thing per crossing.

The kernel's government is a good one exactly insofar as its citizens minimize their petitions, and the next chapter's subject, the manufactured life that makes those petitions, will show where the requests come from and who answers for them.

The kernel's own anatomy deserves a paragraph, because the word covers three distinct citizens that the working professional should stop flattening into one. The first citizen is the interrupt handler, the kernel's reflex: a short, urgent, privileged routine that runs the instant hardware taps, forbidden from sleeping, forbidden from waiting, and confined to recording the event and scheduling the real work for later, since a reflex that lingers starves the whole body. The second is the kernel thread, the kernel's musculature: the deferred work queued by the reflexes is performed by these long-lived privileged workers, which sleep and wake like any process but live on the government's side of the border, tending buffers, flushing caches, and balancing the machine's books.

The third citizen is the driver, the kernel's embassy to each device, translating the kernel's general requests into a particular device's private commands, and holding, uniquely among the kernel's parts, the power to bring the whole government down, since it runs privileged against real hardware with the vendor's authority and no one's supervision. The anatomy explains the machine's political weather: an overloaded interrupt path shows up as latency nobody can attribute, a stuck kernel thread shows up as a bookkeeping queue

that never drains, and a misbehaving driver shows up as anything at all, which is why the oldest diagnostic question in operations, what changed in the kernel's roster lately, remains the best one, and why the honest operator treats the third citizen with the respect owed to a strong ally who is also the leading suspect.

The chapter's final subject is a distinction the industry blurs constantly and the working craft depends on: the difference between the kernel and the operating system. The kernel is the government proper, the resident of the privileged mode, small, austere, and eternal; the operating system is the whole civilization, the kernel plus its civil service, the shells and utilities, the libraries, the package managers, the services that start at boot and speak its conventions, all the machinery that turns a governed machine into a usable one. The blurring matters because the failure modes live at different addresses: a kernel bug is a crash or a hang of everything at once, a civil service bug is a broken utility, a missing tool, a misconfigured service, an annoyance, and the operator who blames the government for a pothole, or the streets for a coup, will misdiagnose in both directions with perfect confidence.

The professional's vocabulary discipline is therefore a diagnostic instrument: say kernel when meaning the privileged resident, say system when meaning the whole civilization, and the bug reports, the postmortems, and the searches of the trade will all get sharper, because the words were load-bearing all along, and the craft of this book, on this layer as on every layer it visits, begins with calling each organ by its true name.

A working kernel exposes a small public square that the professional should learn to read, and its instruments deserve a paragraph each. The first is the load average, the census of the impatient, counting the processes running now plus those ready but unseated; the number is a queue length, not a percentage, so its meaning is always relative to the machine's seat count, and the honest reading is its trend rather than its instant, since a steady queue is a machine doing its business while a climbing queue is a machine falling behind its arrivals.

The second instrument is the accounting of where the machine's seconds actually go, split among the user's arithmetic, the government's bookkeeping, and the waiting, the time the machine spends with nothing to run because everything is blocked on replies from the disk or the wire, and the split turns the vague complaint that the machine is slow into the three distinct diseases the earlier chapters taught: busy is a workload problem, bookkeeping-heavy is a crossing problem, and waiting-heavy is a reply problem, each with its own cure, and the folklore that treats all three with the same reflex, more

hardware, is the industry's most expensive reflex of all.

The third instrument is the kernel's own log, the ring where the government writes what it could not help noticing: the throttled device, the corrected error, the nearly exhausted supply, the process it had to evict, and the professional's daily discipline of reading the ring, even on healthy days, is the quiet habit that turns the government's early warnings into maintenance windows instead of incidents, since the kernel, of all the organs in this book, is the one that complains earliest, in the smallest voice, on the most days.

The paragraph that closes the chapter should also open its political dimension, because the kernel is not only an engineering artifact but a standing argument about who may do what, and the professional lives inside the argument's current settlement. Every capability the kernel grants is a boundary someone drew: the permission to bind a low port, to open a device, to raise priority, to lock memory against exile, to trace another process, each of these is a policy, tuned by decades of practice into the defaults the working day inherits, and every default is simultaneously a security posture and an operational convenience, traded against each other by people the operator never met.

The craft's working stance is neither to worship the defaults nor to fight them, but to know them: the hardened service is the one whose granted capabilities fit its job description and nothing more, the audited machine is the one whose grants are listed, and the incident that ends with the discovery that the attacker's power was, in fact, a granted capability nobody remembered granting is the argument's standard tuition. The reader who finishes this chapter with the kernel as government, market of capabilities, and earliest complainer has the layer's working model in full, and every later chapter, each of which depends on the kernel's borders, will invoke this one without saying so, which is how a book's first organ earns its keep.

The chapter's last working lens is the kernel's behavior under starvation of its own attention, because a machine can run out of government before it runs out of anything else. Every interrupt, every petition, every crossing is served by a shared supply of processor attention, and the supply, like all supplies in this book, is finite and scheduled; a machine whose devices interrupt faster than the kernel can retire their work builds a queue of unretired events, and the queue's depth is invisible in the usual census, since the interrupted processes show up merely as odd latency, their seconds charged to no one.

The signature is the machine whose tools disagree: the load is modest, the processes are sleeping, the logs are quiet, and yet the disk answers late, the wire lags, the clock drifts against its source, each device politely starving the others at the door of a government too busy to notice its own queue. The

professional's discipline is the attention audit: on a machine that serves I/O in anger, watch the counts of interrupts served and work deferred, and treat a growing gap as early weather, since the kernel's attention, unlike its memory, is not lent on demand but rationed by design, and the ration can be exhausted by the devices it governs, which is the political irony of this organ, that the government can be quietly besieged by its own citizens, each of them individually behaving exactly as designed.

One more honesty belongs beside that irony, because the kernel's rationing explains a working category the folklore mislabels: the machine that is fast at the wrong times. A kernel that prioritizes interactive responsiveness, keeping the terminal crisp while the batch work waits, will make the nightly batch feel slow while the daytime operator feels quick, and a kernel tuned the other way, batching aggressively for throughput, will make the daytime operator feel the machine lag while the metrics report magnificent throughput, and both machines are honest, both are correct, and both are misconfigured for somebody.

The craft's lesson is that responsiveness and throughput are competing goods, priced against each other by policy that somebody must own, and the professional who inherits a tuning decision without its policy should treat the tuning as an artifact, test it against the workload the machine actually serves today, and re-price it deliberately rather than preserve it out of respect for a stranger's default, since the stranger's workload is not among the machine's surviving facts, and defaults preserved past their workloads are the kernel-layer version of the long-run chapter's fossil, honored for reasons nobody can name, and priced at the pager, like all the other ones.

The kernel is patched while it lives, and the patching ceremony earns separate treatment here, being the working day's least comfortable maintenance. A user program can be stopped, replaced, and restarted at will, but the government's replacement is the whole machine's interruption: the kernel patch requires a reboot, a boot that takes the machine's services off the wire, and the industry's entire slow war against that interruption, the live patching of small fixes, the careful sequencing of restarts across a fleet, the maintenance windows negotiated against traffic, is the acknowledgment that the government cannot be replaced the way citizens can.

The craft's discipline is the reboot budget: know, per machine and per fleet, how often the kernel's honesty requires a restart, and schedule the budget deliberately, since the machine that never reboots is either a machine with a boring kernel or a machine accumulating the risk of a long-delayed interruption, and the folklore's pride in multi-year uptimes was always the

wrong pride, since the unpatched veteran is a security posture that the long-run chapter will price at its true discount.

Beneath the boot that every patch eventually requires sits the boot's own honesty, the chain of trust, and the paragraph states it plainly because the working professional inherits it silently. The machine's first instructions come from firmware, small permanent programs baked into the hardware's vendors, and the modern boot asks each link in the chain, firmware, loader, kernel, to verify the next link's identity before handing over the machine, a ceremony designed to answer a narrow question, whether anything in the chain has been replaced by a stranger, and the answer's strictness is a policy: a machine that refuses to boot any unsigned kernel is a machine that protects against a class of attack and also refuses the operator's own custom kernels, and the loosened setting trades the protection for the convenience, in the standing pattern.

The professional's duty is to know which policy the fleet's machines hold, since the boot policy is invisible in every daily operation and decisive in exactly one scenario, so that the boot, now visible as a ceremony with a politics, rather than a few seconds of waiting, holds the kernel chapter's last face, the government's own question of who is allowed to govern, asked by the machine, at every start, forever.

A last instrument of the government rounds out the chapter: the trace of the kernel itself, recorded for the whole machine rather than one process. Where the process trace of the second chapter watches a single life, the kernel trace watches the state, recording the machine's moments of interest, a scheduling delay beyond bounds, a blockage that lasted too long, a migration of the working set that forced an exile storm, and the recording is cheap enough to leave running, the professional's habit of keeping the machine's slow moments written down before they become failures, so that the stall reported at noon can be read against the machine's diary from the hour before, when the evidence was fresh.

The discipline that completes the layer's kit is therefore the standing record: baselines of the machine's normal, captured in the quiet hours, kept where the incident will look for them, since the honest diagnostic is always comparative, the machine against its own calm, and the professional who has the calm on file holds the comparison in advance, which is the difference between reading a stall and reconstructing one, and its price is denominated in the one currency the incident charges, which is the hour nobody has.

End of free sample. The full book is available at putrabooks.github.io

ABOUT THE AUTHOR

Aiman JoeHanz writes about infrastructure: the physical and logical systems that carry modern life and the people who keep them running. The author's method is the working anatomy, books that describe no named company, cite no product, and honor no doctrine, only the patterns that survive every generation of tools, and the author's subject is always the same underneath its costumes: the trades that strangers make with strangers so that the lights stay on, the water stays clean, and the machines answer. The author lives with the consequences of every abstraction described in this volume, writes on machines governed by every kernel described here, and offers this book in the series' standing spirit, as a working tool rather than a doctrine, to every professional who tends the invisible middle.