SAMPLE

Building the Cloud Services Infrastructure of Tomorrow

Building Scalable Cloud Systems for Tomorrow

Aiman JoeHanz

Free sample: Introduction & Chapter 1

putrabooks.github.io

Buy the full book:

Carousell carousell.sg/p/building-the-cloud-services-infrastructure-of-tomorrow-ebook-pdf-1467067970

Gumroad

Ko-fi

ElevenReader

Payhip

Contents

Introduction

Chapter 1: The Utility Turn — What Cloud Services Infrastructure Actually Is

Chapter 2: The Layered Model — Abstraction and the Shared Responsibility Bargain

Chapter 4: Compute Primitives and the Economics of Matching Workloads

Chapter 5: Storage and Data Services — The Gravity at the Center

Chapter 6: Networking and Identity — The Perimeter Rebuilt

Chapter 7: Managed Platforms and the Abstraction Bargain

Chapter 8: Reliability — Trust as a Designed Property

Chapter 10: The Road Ahead — Operating the Estate of Tomorrow

Frequently Asked Questions

Glossary

About this Series

Connect

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

Introduction

For most of the history of computing, if you wanted software to run, you bought machines. You housed them, powered them, cooled them, cabled them, patched them, and eventually hauled them away. The machine was the unit of ownership, and everything about information technology organized itself around that fact: the capital budgets, the procurement cycles, the data centers in the basement, the operations teams whose job was to keep physical objects alive. The software was the point; the hardware was the burden you accepted to run it.

Cloud services inverted that arrangement. Instead of buying machines, you rent outcomes: a place to run code, a place to keep data, a network to connect them, an identity system to say who may touch either. The unit of consumption shrank from the server to the hour, the minute, and eventually the single request. Capacity stopped being a building you filled over eighteen months and became a dial you turn in an afternoon. And with that inversion, a new engineering discipline emerged — one that this book calls cloud services infrastructure: the deliberate design of compute, storage, network, identity, and managed platforms, assembled into an estate that serves an organization's workloads reliably, securely, and at a cost that scales with use rather than with ambition.

That discipline is what this book is about. It is not a book about any single provider's console, because consoles change every few months and the menus say nothing about why one design outlives another. It is a book about the durable engineering underneath all of them: the failure-domain thinking that makes systems survive bad nights, the abstraction bargains that trade control for leverage, the economics that quietly decide whether a migration saves money or merely relocates the spending, and the operating habits that separate estates that compound in value from estates that quietly rot.

The word "infrastructure" is doing real work in the title. A service is something you call; infrastructure is something you depend on. The moment an organization's billing, customer records, or product itself runs in the cloud, the cloud estate stops being a convenience and becomes load-bearing structure — as consequential as the electrical supply, and considerably less forgiving of improvisation. Treating it as anything less is the root cause of most cloud failures: unowned architectures, accidental complexity, security shaped like a fence around a shapeless interior, and bills that no one can explain because no one designed them.

Infrastructure also implies a standard of thinking: failure modes considered in

advance, capacity planned rather than discovered, changes reviewed and reversible, and a bias toward boring, well-understood components arranged deliberately over clever components arranged hopefully. The cloud rewards that thinking more than the old world did, because the cloud is programmable — but it punishes its absence faster too, because a misconfiguration propagates in seconds to systems that thousands of people are using.

Most cloud material crowds the two poles. One pole is the provider tutorial: focused, practical, and obsolete by design, teaching where the buttons are without teaching what the machine actually does. At the other end sits the certification curriculum, which enumerates every service in the catalog and the shape of every exam question, but rarely confronts the decisions that actually ruin projects: which abstraction to accept and which to refuse, how much redundancy is enough and for which failures, when the platform's convenience becomes a trap, and how to tell the difference between a system that is cheap and a system that merely looks cheap.

Between those extremes sits the engineering judgment, and that judgment is this book's territory. Each chapter takes one structural dimension of cloud services infrastructure — the layering of the stack, the geography of regions, the economics of compute, the gravity of data, the shape of the network, the bargains of managed platforms, the design of reliability, the discipline of security and cost, and the trajectory of the field — and treats it as infrastructure: what it really does, why it is built the way it is, how it fails, and how to design with it rather than merely on it.

The chapters follow the subject's own structure rather than a syllabus's. Chapters 1 through 3 build the model: the utility turn, the layered stack, and the geography of failure domains — the vocabulary and the physics that everything else assumes. Chapters 4 through 7 are the estate's material science: compute, storage, networking, and the managed-platform bargains that bind them. Chapters 8 and 9 are the operating disciplines — reliability, security, and cost — that decide whether the material becomes a structure or a pile. Chapter 10 looks ahead and assembles the whole into a build sequence. Readers who operate estates daily may prefer to begin with the discipline chapters and work backward to the foundations; newcomers should take them in sequence, since the later chapters lean on vocabulary the earlier ones built.

A note on claims: statistics, benchmarks, and provider feature lists are left out on purpose. Figures decay, and the providers' own documentation catalogs them faster than print ever will. The book trades them for the durable part — the trade-offs, named and priced, nearly invariant across vendors and years.

When a statement is the author's judgment, it says so; where practitioners still

argue, the argument itself is presented instead of a verdict being imposed.

First, the engineer or architect who designs, builds, or operates in the cloud and wants the machine beneath the console understood — the person who suspects, correctly, that durability comes from understanding, not tooling. Second, the technology leader who approves, budgets, or audits cloud work and needs the questions that separate solid estates from well-marketed ones. Third, the student or career-changer just arriving, for whom these pages try to be the layer certification curricula skip: not what the services are called, but why the well-built ones were built the way they were. None of it requires a specific provider, and all of it survives the next redesign of every console.

Chapter 1 establishes the foundation: what cloud services infrastructure is, how it differs from owning machines, and why the utility model changed which decisions matter. Chapter 2 walks the layered model of the stack — from concrete floors to managed platforms — and the shared-responsibility bargain that defines what a cloud user actually owns. Chapter 3 treats geography: regions, availability zones, latency, and the physics of failure domains. Chapter 4 examines the compute primitives and the economics of matching workloads to them. Chapter 5 turns to storage and data services, including data gravity and the estate that grows around a database.

Chapter 6 covers networking and the new perimeter built on identity rather than location. Chapter 7 confronts the managed-platform bargain: what abstraction buys, what it costs, and how portability is really engineered. Chapter 8 treats reliability as a designed property — objectives, error budgets, disaster recovery, and dependency hygiene. Chapter 9 binds security, governance, and cost into one operating discipline. Chapter 10 looks ahead at edge computing, sovereign clouds, and the capacity pressures of AI workloads, and closes with a build sequence for organizations assembling their own estate.

The approach throughout is evergreen: no step-by-step console instructions, no statistics that will be stale by the time you read this, and no provider marketing wearing the costume of analysis. Just the engineering, described plainly, the way a colleague would explain it on the way to a design review.

If you are responsible for designing, operating, approving, or auditing systems that run in the cloud — as an engineer, an architect, a technology leader, or a student of the field — the goal of this book is to leave you with the judgment layer: the questions to ask, the trade-offs to weigh, and the reasons the well-built estates you encounter were built the way they were.

A closing note on stance: infrastructure engineering is an optimistic discipline.

It assumes that systems can be made dependable by design, that failures can be made boring by rehearsal, and that complexity can be made legible by writing it down. The cloud gives that optimism more to work with than any era before it — the challenge, and the pleasure, is being worthy of the machinery.

Chapter 1: The Utility Turn — What Cloud Services Infrastructure Actually Is

Every technology wave announces itself with a change in vocabulary, and the cloud was no exception. Words like "data center" and "server room" gave way to "region," "instance," "bucket," and "endpoint." The vocabulary shift was not cosmetic. It marked a genuine inversion in what an organization owns, what it operates, and what it can change. Understanding that inversion precisely — not as slogan, but as engineering and economics — is the foundation on which every later chapter of this book rests.

This chapter defines cloud services infrastructure, traces how it differs from the owned-machine model that preceded it, explains why the utility comparison is both apt and limited, and sets out the four properties — elasticity, programmability, shared tenancy, and consumption pricing — that make the cloud a different kind of thing to engineer rather than merely a different place to put servers.

Under the owned-machine model, the fundamental object was the computer. An organization bought hardware, installed an operating system, and then coaxed applications onto it. Capacity was a physical stock: if a system needed to handle twice the traffic, the answer involved purchasing, racking, cabling, and burning-in new machines — a process measured in weeks or months. The consequence was a permanent tension between over-provisioning and embarrassment. Buy too much and capital sat idle in a room that cost money to cool. Buy too little and the system failed exactly when it mattered most, on the day everyone showed up.

The cloud model relocates the machines to someone else's buildings and shrinks the unit of consumption to something a software call can create or destroy. You do not rent a computer; you rent compute — the effect of a computer, delivered as a service and metered by the second, the hour, or the request. The same is true of storage, of network transit, of database engines, of message queues, of the identity system that decides who may use any of it. Each of these was once a thing you built and cared for. Each is now an outcome you request.

That shift sounds modest and is not. When the unit of consumption is the request rather than the rack, capacity planning stops being a procurement problem and becomes a software problem. When every resource is created by calling an interface, the entire estate becomes describable, reviewable, and reproducible in code. When thousands of tenants share the same physical substrate, economics that were previously impossible — paying for a fraction

of a top-of-rack switch, or a slice of a machine too powerful for any one tenant's need — become the default. Each of these consequences gets its own treatment later in this book; here the point is that they all flow from the same root: the object of engineering changed.

This book uses a working definition: cloud services infrastructure is the deliberately designed assembly of compute, storage, network, identity, and managed platform services — rented from a provider, but architected, owned, and operated by the organization using them — that together serve an organization's workloads.

Two words in that definition deserve emphasis. The first is "deliberately." Nothing about renting services forces good design. It is entirely possible to consume cloud services the way an unsupervised customer consumes a supermarket — impulsively, without a list, and with a bill that surprises everyone. An estate is cloud services consumed deliberately: with an intended shape, known failure modes, and someone accountable for each layer. The second word is "assembly." The cloud provider sells parts and primitives. It does not sell architecture. The architecture — which primitives, in which combination, with which failure assumptions and which boundaries — remains the customer's work, and the quality of that work is the difference between an estate and an accident.

This is why the phrase "we moved to the cloud" often disappoints. Moving is not designing. An organization that lifts its existing servers into rented virtual machines has changed its invoices but not its engineering, and the monthly revelation of the new bill is frequently the first moment it discovers the difference.

The cloud is routinely compared to the electrical grid, and the comparison earns its popularity: both deliver a standardized commodity at a metered price, both freed their users from capital investment in generation, and both became so reliable that whole industries reorganized around them. The comparison is genuinely useful for one lesson in particular — that infrastructure is only infrastructure when you can stop thinking about it — and genuinely misleading in two ways.

The first misleading part is reliability. The electrical grid delivers power so predictably that a household can ignore its failure modes; the cloud does not.

Providers operate at remarkable quality, but every large cloud region has had bad nights, and the services within a single provider fail independently of one another. The grid hides its failure modes from the household; the cloud surfaces its failure modes directly into the customer's architecture, in the form

of regions that must be chosen, availability zones that must be spread across, and service-level agreements that must be read with contractual care. The cloud hands reliability back to the customer as a design parameter, which is a gift and a burden at once.

The second misleading part is interchangeability. Electricity is a commodity with a plug standard; one electron is like another, and switching suppliers is a metering change. Cloud services are not like this. Each provider's managed services, identity systems, and primitives carry their own semantics, and an application built deeply into one provider's conveniences does not move to another the way a lamp moves between sockets. Portability is real, but it is an engineering property you design for, not a property the market gives you.

Chapter 7 returns to this at length.

Everything the cloud changes flows from four properties, and they are worth naming exactly, because each one converts an old operating habit into a new one.

Elasticity. Capacity can grow and shrink with demand, on timescales from minutes to seconds. Elasticity converts the old stock problem — how many machines do we own — into a flow problem: how does the system scale out, and how does it scale back in. The second half of that sentence is the half organizations forget. A system that scales up under load but never scales back down is elastic in the way a ratchet is: it moves in one direction, and only the bill notices.

Programmability. Every resource is creatable, modifiable, and destroyable through an interface, by code. This is the quiet superpower of the model, and it is larger than it first appears. Once the estate is created by code, the estate can be versioned, reviewed, tested, reproduced, and audited like any other software artifact. The industry term — infrastructure as code — names a practice this book treats as non- optional, and Chapter 9 makes the case that an estate that cannot be rebuilt from its code is not owned, merely occupied.

Shared tenancy. Many customers share the same physical plant, and the economics of scale that result — fractions of machines, region-sized networks, security teams the size of small companies — are passed down as capabilities no individual tenant could afford alone. The costs of sharing, isolation engineering chief among them, are borne by the provider, but the residual risk of sharing is borne by the tenant, and honest engineering acknowledges both directions of that ledger.

Consumption pricing. Cost tracks use rather than intent. The discipline this forces is healthy and rude: in the owned world, waste was invisible, sunk in the purchase; in the cloud, waste appears monthly, in an invoice, with the name of the team responsible for it. Consumption pricing turns cost from a capital event into an operating signal — one that should, in a well-run estate, feed straight back into design.

The utility turn changes the economics of capacity and the mechanics of operation, but it quietly preserves the hard parts of engineering, and pretending otherwise is where disappointment begins. Software still has bugs. Dependencies still fail. Capacity still has to be planned, only at the level of services and limits rather than racks and rooms. Security still depends on knowing exactly what exists and who may touch it — a problem that becomes harder, not easier, when creation is one API call away. And judgment is still

scarce. The cloud removes none of the questions this book is about; it re-poses them with faster feedback and larger blast radius.

The chapters that follow walk those questions in order: the layered shape of the stack and the responsibility it assigns, the geography that contains it, the economics that drive it, the data that anchors it, the network and identity that guard it, the platforms that accelerate and encumber it, and the reliability, security, and cost disciplines that keep it honest. The place to begin is the stack itself — because nearly every failure in cloud engineering is, at root, a misunderstanding about layers.

It is worth being candid about the provider landscape, in a way that survives rebranding and consolidation. A handful of hyperscale providers operate global fleets and set the pace of the industry; their scale lets them price raw capacity aggressively and push managed innovations quickly. A larger set of providers competes on specialization: regional presence, regulatory fit, price on specific workloads, or service bundles attached to other business relationships. Below both sits the platform layer — the software that makes clouds out of hardware — which has made it possible for anyone with a data center to sell a cloud-shaped experience.

For an engineer, the differences that matter are rarely the marketing ones. They are, in rough order of consequence: the depth and coherence of the managed-service catalog; the quality and granularity of identity and permission systems; the maturity of infrastructure-as-code tooling; the shape of the network primitives; the clarity of the pricing model; and the specifics of the service-level commitments. Those six axes do more to determine what an architecture will look like than any feature announcement, and they change far more slowly than the surface of the product.

The choice among providers is therefore not the choice this book spends its energy on. Architectural judgment transfers across providers far better than console familiarity does, and the chapters ahead are written to that transferable layer: the model of the stack, the physics of failure, the economics of consumption. Where provider specifics illustrate a principle, the principle is the point.

The utility turn is also a change in how money works, and the change is easy to state and easy to mishandle. Under the owned model, spending concentrated in capital purchases followed by long tails of operating cost: power, cooling, staff, maintenance contracts, and the silent costs of obsolescence. Under the cloud model, almost all of it becomes operating expense, paid in proportion to use. Finance departments call this a shift from capital expenditure to operating expenditure; engineers should call it a shift

from buying capacity to renting outcomes, and should notice what it does to decision-making.

In the owned world, the punishment for a bad capacity guess arrived once, at purchase, and then faded. In the cloud, the punishment for a bad design decision arrives every month, without fading, until someone fixes it. This is why cloud economics so often feels worse than the old world even when it is better: the old world hid its waste in sunk capital, while the cloud itemizes waste with uncomfortable precision. An estate that costs less than the data center it replaced can still feel more expensive, because every dollar of it is now visible, attributable, and disputable. The correct response to that visibility is not nostalgia but exploitation — treating the bill as the highest-frequency design review the organization receives, a subject Chapter 9 develops fully.

A quieter property of the cloud deserves its own moment, because it drives progress in the field more than any single product does: the cloud standardizes by demonstration. When a provider makes a managed database, a message queue, or an identity directory available to every tenant at once, it effectively publishes a reference implementation of that component. Tenants adopt it; tooling grows around it; the industry's collective idea of what a database or a queue should do recalibrates toward the managed shape. Over time this creates a ratchet — the floor of what counts as acceptable keeps rising, because the baseline components available to everyone keep improving.

The ratchet cuts both ways. It lifts everyone's floor: capabilities that once demanded specialist teams — replicated databases, globally distributed storage, hardware-backed key management — become an afternoon's work. But it also compresses the middle: the engineer whose expertise lies in operating, say, a specific self-managed middleware loses leverage every year, because the platform absorbs that expertise into a service with a checkbox. The durable skills are the ones above the ratchet — judgment about when to accept the platform's answer, how to compose it, and how to verify it — and those are precisely the skills this book teaches.

The final foundational skill is learning to read a cloud service catalog the way an engineer reads a datasheet rather than the way a shopper reads a brochure. Every service in the catalog is an answer to a question, and the catalog is enormous because the questions are. The productive habit is to translate each service back into its underlying question: object storage is "where do I put bytes I rarely move"; a managed orchestrator offering is "who patches and scales my container runtime"; a serverless function is "how do I run code on an event without owning the loop"; a secrets manager is "where do credentials live so they are neither in code nor in people's memories."

Read this way, the catalog stops being intimidating and becomes a menu of pre-negotiated answers. The engineering work is the questions in between: which answers to compose, which to refuse, and where the composition itself — not any single service — becomes the thing your organization actually owns. That compositional layer is where every remaining chapter of this book lives.

The model this chapter has described did not arrive as a blueprint; it accumulated. Its first act was remote machines — the same servers, owned by someone else, rented by the hour, which is why the earliest cloud migrations were, honestly, relocations. The second act was the primitives gaining programmatic depth — the networks, disks, and images of Chapter 2 becoming first-class API objects, which made the estate describable in code and set up everything this book means by infrastructure as code. The third act was the managed services — the databases, queues, and platforms of Chapter 7 — which moved the bargain from renting machines to renting operating expertise. The fourth act, still arriving, is the platform consolidation and the AI-shaped capacity of Chapter 10.

The history matters for one reason: each act raised the floor of what every organization gets by default, and each act widened the gap between the organizations that understand the whole model and the ones that only use its newest layer. The engineer who joined during act three and never learned act one's virtual machines will design serverless systems without understanding what they run on; the manager who joined during act one and never internalized act three will keep renting fleets to avoid the bargains that would save the budget. This book walks all four acts deliberately, because the estate of the next decade will be built by people fluent in all of them.

Because the rest of the book uses a handful of words with deliberate precision, the definitions are collected here, once, plainly. The estate is the whole of what an organization runs in the cloud: its resources, its rules, and its operators. A primitive is a raw building block — a machine, a disk, a network — that the tenant shapes into whatever it needs. A managed service is a component the provider operates as a product. A workload is a system with a job to do — the checkout service, the nightly pipeline — and the estate exists to serve workloads, not the reverse. A tenant is one customer of a shared platform: you. A failure domain is a set of things that a single failure can take with it, from one machine to one provider. These words recur in every chapter because the book's subject is the relationships among them, and the relationships — not the vocabulary — are the engineering.

One property of the utility model deserves to be named before the book

descends into layers, because it explains the subject's reputation for difficulty. The cloud's parts are individually simple — a machine here, a bucket there — and the estate's complexity comes not from any part but from composition: the enormous number of combinations the parts permit, and the fact that nothing in the model prevents a bad combination. The old world constrained design by scarcity — machines were expensive, so architectures stayed small; the cloud removes the constraint and, with it, the guardrail. What replaces the guardrail is judgment, and judgment is what the remaining chapters are for: the layer model next, then geography, then the material science of compute and data, then the bargains, then the disciplines. The parts of the cloud are a menu; the engineering is the meal.

Every field is built on one mistake its practitioners must unlearn, and cloud engineering's is this: the belief that the cloud's flexibility removes the need for design. The flexibility is real — anything can be created in minutes, changed in an afternoon, undone in a moment — and the unlearning is that these are all the same fact: with the guardrails of scarcity gone, the cost of a bad decision is no longer paid at design time, when it is cheap, but at operation time, when it is not. The old world's constraints forced deliberation; the cloud's abundance merely hides the bills and the breaches until they are invoices and incidents. The engineering answer is not nostalgia for scarcity; it is bringing the deliberation voluntarily. The chapters ahead are that deliberation, organized.

One boundary of the utility model is drawn incorrectly so often that it belongs in this chapter rather than a footnote: the belief that adopting the cloud outsources responsibility along with the machines. What transferred to the providers is concrete: buildings, hardware, the operating of platform software. What never transferred, under any provider or any service level, is accountability for the estate — the answer to the questions of what exists, what it holds, who may act, and what it costs. That answer has exactly one home, the organization that benefits from the estate, and every service agreement in the catalog says so in its own way. The utility turn, properly understood, is not a transfer of responsibility but a concentration of it: the cloud removes the excuses — the slow procurement, the unavailable skills, the full basement — and leaves the organization alone with its decisions. The rest of this book is about making those decisions well.

One more consequence of the utility turn deserves a foundation-level mention, because it explains much of the confusion in the field: the cloud changed the shape of the engineering career itself. Under the owned-machine model, operations and development were separate tribes — one tribe wrote software, the other kept machines alive, and they communicated through tickets. When capacity became an API, that boundary dissolved. The people who write

software now design the infrastructure it runs on, and the people who understand infrastructure write the software that provisions it.

The result is a discipline that is neither pure development nor pure operations, but the synthesis the industry calls platform engineering: the practice of building the estate — and the paved roads, templates, and guardrails around it — as a product that other engineers consume. This book is written for that synthesized discipline. The reader it assumes is comfortable writing code, but the code the reader will be judged on is the code that describes and operates the estate: templates that provision infrastructure, pipelines that deploy it, tests that validate it, and monitors that watch it. The chapters ahead treat those artifacts as first-class engineering deliverables, because that is what they are.

The first and most consequential of those deliverables is an accurate model of the stack itself — what sits on what, what each layer promises, and where the responsibility for each layer lands. That model is the subject of the next chapter.

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

ABOUT THE AUTHOR

Aiman JoeHanz studies the systems the modern world runs on and writes them down plainly. His Building Infrastructure of Tomorrow series treats each layer of the technology stack as engineering -- finance, government, healthcare, operations, talent -- and this seventh volume turns to the rented substrate underneath nearly all of it: cloud services. He writes for practitioners and the leaders who approve their work, favoring durable trade-offs over feature lists.

He lives somewhere between a terminal and a design review.