


THE ENGINEER'S SURVIVAL GUIDE

Expert advice for handling
workload (and work-life) disasters



Written by Michelle Gienow
Illustrated by Giovanni Cruz





“Software engineering today is a race between software engineers striving to build bigger and better idiot-proof apps, and the universe striving to produce bigger and better idiots. So far, the universe is winning.”

– Rick Cook



THE ENGINEER'S SURVIVAL GUIDE

**Expert advice for handling
workload (and work-life) disasters**

Written by **Michelle Gienow**

Illustrated by **Giovanni Cruz**

 **Cockroach Labs**

The Engineer's Survival Guide: Expert advice for handling workload (and work-life) disasters

Written by **Michelle Gienow**

Illustrated by **Giovanni Cruz**

Copyright © 2023 Cockroach Labs, Inc.

Published by Cockroach Labs, 125 25th Street, 11th Fl, New York, NY 10001

www.cockroachlabs.com

Editors: Jessica Edwards, Dan Kelly

Copyeditors: Alex Kimball, Steven Lichtenstein

Contributors: Dan Kelly, Andrew Marshall

Project manager: Irina Sapsay

Designer: Giovanni Cruz

Legal counsel: Ruchi Shah

October 2023: First edition

The CockroachDB and Cockroach Labs logos are registered trademarks of Cockroach Labs, Inc. The Engineer's Survival Guide: Expert advice for handling workload (and work-life) disasters, the cover image, and related trade dress are trademarks of Cockroach Labs, Inc.

The views expressed in this work are those of the author and do not represent the publisher's views. While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and rights.



WARNING

Are your hands tangled in back-end spaghetti courtesy of the startup you acquired? Did management really just tell you to use AI instead of hiring a human? WHAT DO YOU MEAN WE'VE BEEN FINED FOR VIOLATING EUROPEAN DATA PRIVACY LAWS — WE DON'T EVEN DO BUSINESS THERE!!!

When a dire situation is at hand, sensible solutions may not be immediately apparent. Because sh*t happens (and, at scale, sh*t is *always* happening) we asked subject matter experts for guidance. Whether the disaster response is directed confidently from across your desk or while hiding underneath it, this book provides an initial roadmap for what to do in a given moment of crisis. The details of *how* to do it, though, are entirely up to you. Every organization and every application are different, and we wouldn't dream of dictating your workloads or your work life.

To deal with the worst-case scenarios presented in this book, we highly recommend (actually, we insist) that you carefully evaluate the situation before you act, and that you act within the boundaries of legality, vendor agreements, and physics itself. Breaking your app is one thing; breaking the law is another thing entirely.

Here's the part our Legal team makes us say out loud: the authors, the experts, and Cockroach Labs do not have any liability for any harm or injury (physical or mental) that could occur by using the information in this guide. We do not claim that the guidance offered in this book is complete or accurate for your specific situations. Moreover, it should never supersede your own judgment and common sense — which can be a challenge in crisis moments when stakeholders and incident management teams are running around screaming and everything is on fire and you worry that all the engineering blood, sweat, and tears in the world won't be enough to extinguish the <clusterfail>.

Stay calm. If you can keep your head when everyone around you is losing theirs, you and your app will survive to deploy another day.

#Acknowledgments

Many technical brains were picked to produce the guidance presented in this, well, guide. Senior subject matter experts from all across the tech sector and all around the planet contributed their insights and lessons learned from surviving their own technical disasters. Names have been redacted to protect the innocent (and to protect the author from years of putting requests through Corp Comms permission chains).

Director of Global Infrastructure, Fortune 50 Financial Services
Mysterious monoliths frighten me.

Senior Cloud Architect, SaaS IoT
*Tired of playing babysitter to Sh*tOps.*

Director of Distributed Tech Ops, Retail
Been there. Broke that.

VP of Application Development, Gaming
That is one crazy idea. Let's do a PoC.

Senior Engineer, Aviation & Aerospace
It works in my container _(ツ)_/

SVP & Head of Technology, Retail
No, I can't fix the printer. I'm an engineer, not a magician.

Principal Cloud Architect, Logistics
JUST >SUDO IT

Application Modernization Engineer, IaaS / Cloud Service Provider
I didn't choose the bug life. It chose me.

TABLE OF CONTENTS

Chapter 1: Surviving your job (#HugOps)	10
What to do when your CEO wants to “hire” a bot	11
How to bail a developer out of jail	15
How to survive a delusional delivery date	19
How to survive a swarm of Swifties	23
How to dig out of (someone else’s) technical debt	27
How to indulge in free snacks and still fit into your work pants	31
True tales of survival: 20 years of server solitude	34
Chapter 2: Surviving the workplace	35
How to survive a “blameless post-mortem” when it’s actually your fault	36
What to do when your rubber ducky debugging buddy goes missing	39
How to help your team break up with bad tech	42
How to lure your team back into the office without inciting a mutiny	46
How to survive a meeting that should have been an email	51
True tales of survival: Surviving a hurricane with a bucket brigade	55
Chapter 3: Surviving the future	57
How to survive a Kaiju attack on your data center	58
How to save your company \$1 billion (no, really)	61
How to protect your ass(ets) from AI attacks	65
How to duct tape a shattered single pane of glass	69
How to build an emergency life support system for a legacy app	73
How to justify blowing up your tech stack while playing a round of golf	76

CHAPTER




SURVIVING YOUR JOB (#HUGOPS)



Them: It's not DNS. There is no way it's DNS.

Narrator: It was DNS.

WHAT TO DO WHEN YOUR CEO WANTS TO “HIRE” A BOT

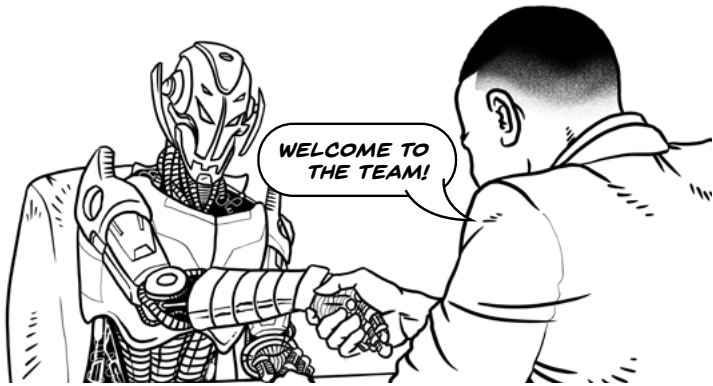


Programmers consult Stack Overflow for help with code so often that it's practically a staff engineer on many projects — which isn't quite the same as making AI *an actual tech team member*. Now, though, your hiring budget's being slashed and your CEO is telling you that ChatGPT is “just as good” as another engineering hire.

Here's how to get useful work out of generative AI tools until you can (hopefully) hire a human.

Treat AI like it's a junior engineer.

AI happens to hold the knowledge of the entire internet. Despite this, generative AI needs explicit and detailed instructions to do anything useful — and its output requires careful review and refactoring, and sometimes even re-refactoring.



In other words, working with AI is just like working with a junior engineer...one that can't accidentally delete the production database.

You'll get the most out of this virtual AI "team member" by using it for tasks that: (1) you would feel confident assigning to a very new or junior engineer; and (2) where it can't screw up anything truly important.

Bring on the boilerplate.

Any grab-a-code-snippet-from-Stack-Overflow instance is exactly when you could ask ChatGPT to generate code instead. Be explicit and give lots of details on what you need the code to do and how you want it done, just as you would with a n00b programmer.

Unlike humans, generative AI typically gets syntax right the first time, which at least saves you time chasing down compile-time and runtime errors.

Also: AI is great at regex. Just saying.

Try, try again.

If your AI-generated code hits any errors, feed them back in as the next prompt. The bot will then tell you how to fix the bug. Some will even apologize nicely for getting it wrong.

This works great in reverse, too: hand the bot any bugs from code you've written and it will return an explanation, plus a (usually) good fix.

Ask it “What does this do?”

Another tedious task perfect for handing off to AI is finding out what a block of code that you didn't write does. This also works for code that you *did* write, intending to come back and comment later. (Narrator: *This never happened.*)

Rather than wasting time puzzling through mystery code, just feed it in and let the bot tell you what it does.

Save time on tricky functions.

Of course you're capable of writing recursive functions or multiple nested loops. But these kinds of problems carry a heavy cognitive load. AI generates solidly average code that you can refactor and modularize as needed in just a few minutes. This gives you more time and focus to invest in architecture, strategy, and business requirements — all places where AI can't provide meaningful help (yet).

Oh, and be sure to ask the bot to take care of the code commenting, too.

Unlike humans, generative AI typically gets syntax right the first time, which at least saves you time chasing down compile-time and runtime errors.

Mind the fine print.

ChatGPT and generative AI tools potentially own the copyright to any code they generate. Are you inappropriately incorporating copyrighted code into your product? Is that a problem? Have you popped into the #Legal Slack channel lately?

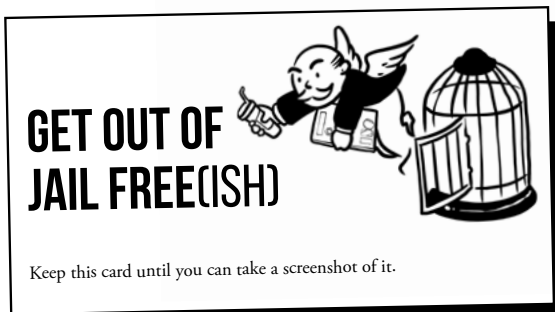
HOW TO BAIL A DEVELOPER OUT OF JAIL



For whatever reason, this person is a human point of failure for a system or service crucial to your business. Maybe you went with “buy” over “build” for a critical solution but then made a single person responsible for integrating and managing it. Maybe this is the last remaining person from the team responsible for manually sharding your database, but now another team has pushed an update that brought everything crashing down, and only the developer knows all the places where the application data logic bodies are buried, figuratively speaking. (Or perhaps even literally. Could that be why they’re in jail?)

Yadda, yadda, hero culture bad, but A Very Important Thing is down and only one human in your organization can fix it fast, if at all. Unfortunately, that human is currently enjoying the hospitality of your local law enforcement agency.

How do you get Bus Factor One out of jail and logged back in to GitHub *right now*?



Find out about arraignment ASAP.

After arrest, a person is either released or jailed. If they're in the slammer (like your Single Point of Failure friend), they'll need to be arraigned before a judge to determine the bail or whether they're released on their own recognizance.

⚠ Warning: On weekends or holidays, it could take several days to schedule arraignment.

Learn their booking status.

No matter what mischief or mischance has landed them in the pokey, your unicorn can't make bail until they've finished going through booking (i.e., fingerprinting, photographing, and intake paperwork).

! Many jurisdictions offer real-time online information on the status of people being held in jail, including location and booking status.

Make a plan for paying bail.

Once bail is set, they can post a bond to leave jail pending trial. You can either pay cash or post bond to spring your jailbird genius. If it's a high-dollar figure (what the heck did they *do!*?), you can make arrangements with a bail bondsperson.

[Optional] Find a bail bondsperson.

There are typically a few 24-hour bail bonds companies surrounding the jail or court in every city. They'll put up money to fulfill the bail requirement, charging a percentage of the total bail as their fee (typically, 10–15 percent).

It's often less expensive overall to just pay the bail up front, if you can manage it. (Getting your developer back to work could qualify as a business expense. Don't forget to ask for a receipt.)

Post bail at the arraignment.

Being able to post bond directly after arraignment is the quickest route to getting your developer out of court and logged back in to GitHub. Bring either the bondsperson or your checkbook.

- i Go plastic:** More courts now accept credit cards and even EFT payments for bail, fees, and fines. Might as well get those airline miles.

Arrange a ride.

Your human point of failure will be released from the holding facility carrying only what they had on them when arrested. They'll appreciate having a way to get home since their phone died hours ago and they can't summon a Lyft.

- i Feed the beast:** If you're taking them directly to the office, definitely make a food stop along the way. They've had a really long day already and, well, the less said about jail food the better.

Start upping your bus factor.

You absolutely need to start a knowledge transfer plan RIGHT NOW. Real talk: this single-point-of-failure situation *should never have happened in the first place*.

The most valuable knowledge often resides in people's heads and is tricky to transfer. It's one thing to know what the code does, but quite another to understand where the hidden trap doors are. Enable live onsite training to cover development, QA, staging, and production environments. Record these sessions and save them in a secure repository for future reference.

HOW TO SURVIVE A DELUSIONAL DELIVERY DATE



Anyone who's been in tech for more than ten minutes has almost certainly been in the unfortunate position of being handed a project with an absurdly short timeline (often accompanied by an absurdly tiny budget). If you haven't, well, *your time will come*.

There's no easy solution to this dilemma, but there are at least a few tactics that can help you survive the experience without hurling your mechanical keyboard through the nearest window.

Don't fight the power.

First, no matter what Chuck D and Flava Flav say, don't fight the assignment or its parameters head on. The likelihood of actually changing any element of this charlie foxtrot equation through direct argument, no matter how logical, is realistically zero. Doing so will (definitely) waste time and (probably) get you labeled as a malcontent.



Make a counter offer.

Use the time you've saved to create an alternative project scenario that's within human capacity to actually pull off.

Begin with some truth telling. Point out that meeting the deadline as is will require more headcount, a bigger budget, and maybe even a contractor or two. Make a strong case and you never know — you might even get a yes. *<Cries in DevOps>*

Now that you have their attention, present your preferred alternative solution:

- ▶ **Option #1:** Same project, only with a more realistic timeline and budget. Explain that, based on your experience and given the resources in place, this is how long you anticipate the project will actually take to complete.
- ▶ **Option #2:** Perhaps an iterative solution would satisfy the need for speed? Suggest deploying a Minimum Viable Product (MVP) to meet the initial deadline, explaining you can then iterate forward until the outcome delights your users – and your bosses. If, after this, the powers continue to insist on a delusional delivery timeline...well, at least you tried.

Obey the laws of physics.

Don't overstress yourself and your team attempting to warp the space / time continuum to meet a ridiculous deadline. The fact that this delivery timeline is untethered to objective reality is not your fault.

- ⚠ **Avoid heroics:** If sustainable, though, occasionally working a few extra hours — especially when milestones are approaching or if any piece of the project seems actually achievable — is a strategy to consider. The goal is to demonstrate *I recognize the deadline and am working to meet it* while not burning out. Slow and

steady (with an occasional curse mumbled under your breath) wins the race.

Beat them at their own game.

Is it possible that leadership is playing a mind game based on Parkinson's Law ("All work expands to fill the time allotted for its completion")? Perhaps setting a ludicrously short deadline is a misguided attempt to spur productivity.

Even if not, they surely anticipate that delays will occur. So when time grows short, they'll throw on another 3–6 months to the deadline without issue.

Perhaps some unhappy noises will be made, but the project will be extended and life will go on.

Don't overstress yourself and your team attempting to warp the space-time continuum to meet a ridiculous deadline. The fact that this delivery timeline is untethered to objective reality is not your fault.

HOW TO SURVIVE A SWARM OF SWIFTIES

In late 2022, Ticketmaster made global headlines (not the good kind) after a major system meltdown when tickets for Taylor Swift's Eras tour went on sale. Response from her fandom — with a population and economic influence large enough to qualify for its own seat in the United Nations — crashed the ticket sales platform under the sheer weight of demand. The backlash was fierce: frustrated fans vented their fury live on social media, members of Congress discussed opening an antitrust investigation into Ticketmaster's alleged monopolistic behavior, and vengeful Swifties quickly filed a class-action lawsuit.



The truth is, this isn't the first large company to experience such a brutally public system failure — and, so long as businesses rely on technology, it certainly won't be the last. The Swifties schooled every business in the world on an important lesson: *You screw up an experience like this, it's dangerous.*

So what can enterprises learn from the TayTay-Ticketmaster meltdown to help prevent their own epic public facepalm?

There's no such thing as “unprecedented demand.”

When a meltdown of this size and seriousness hits, companies typically claim, “There was unprecedented demand on our system!” Sorry, but you're the problem. It's you.

No matter what goods or services your company slings, *your entire reason for existing is to meet customer demand.* So do what you need to meet it.

Scaling during an emergency is not a strategy.

The Ticketmaster disaster (Taylor's version) is but another object lesson in capacity planning: *Don't wait until a surge event to figure out how much traffic is too much traffic.*

Building scalable systems that can handle spiky workloads requires considered planning, not just a vague notion to simply throw money at more servers in the event you unexpectedly hit number one in the App Store.

Choosing inherently scalable application architecture, components, and services requires careful consideration of the infrastructure, caching layers, APIs, database, and more.

- ❗ **Put the right pieces together:** Doing it in the right way will automatically distribute even the largest of loads, allowing your app to simply shake it off.

Uncork bottlenecks.

Common capacity planning means bracing for a threshold of 10x your current or projected peak throughput (*cough*, load testing, *cough*). Ticketmaster reported being “hit with three times the amount of bot traffic than we had ever experienced” during the peak Swiftie Swarm. Assuming the company had 10x planning in place, a failure at 3x indicates their system could have an unrecognized bottleneck that caused cascading failures throughout the system when 14 million users (both humans and bots) all tried to squeeze through the same door at the same time.

⚠️ **Learn from Ticketmaster's pain:** Unrecognized bottlenecks are, of course, A Bad Thing. Plan for, test, and recognize them before they become a problem.

Embrace the chaos (engineering).

Surge events increase the likelihood of any back-end service experiencing a possible degradation or outage for many reasons, including flooded network capacity, spiked CPU, or storage quota exceeded.

Using chaos engineering pushes teams to deal with forced system failures. These unpredictable worst-case scenario drills expose weaknesses (see above) and provide a disaster response experience that helps decrease RTO¹ in the event of a real-world SHTF² event.

Build a time machine.

If a massive outage strikes your product / service / platform before you ever got around to meaningful capacity planning or updating your disaster recovery plan, well...your best (and honestly only) option for preventing the inevitable public outpouring of user outrage would be a sudden breakthrough in time travel technology.

¹ Recovery Time Objective, the goal your organization sets for the maximum length of time it should take to restore normal operations in the event of an outage or data loss.

² A catastrophic event; when the "sh*t hits the fan."

HOW TO DIG OUT OF (SOMEONE ELSE'S) TECHNICAL DEBT



It's the age-old story: you need a new feature so you can continue crushing competitors and keep users. You find a smaller company providing exactly that solution, and they're ripe for acquisition. You both swipe right, expecting to live happily ever after.

The ink is barely dry on the contract, though, when you discover that the seemingly solid tech that passed your code smell testing actually has considerable junk in the technical trunk.

There are millions of possible technical debt nightmares. Some are easily spotted (massive God-class abstractions with names like `driver_manager_system_sub_v242_final_final2_ REALLYFINAL`). The worst come with hidden third-rail boobytraps (touch them and something dies). But all are recognizable by the fact that their continued existence slows you down, hurts development, and ultimately costs the business money.

What's the best way to shave this yak, pronto, and keep your super-fast TTM dreams alive?

Stop adding new stuff.

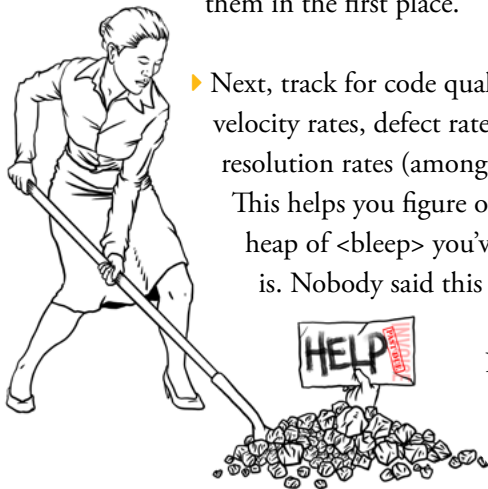
It's tempting to immediately throw technology at the worst / most obvious problems, but more tools aren't going fix the situation until you know what the full situation actually **is** — and how much sauce you're going to need for all that spaghetti.

Make a list, check it twice.

Sing along! *Gotta find out what's naughty or nice...*

- ▶ First, procure an accurate and detailed accounting of current technical debt. That includes document assets, data, and their importance to the reason you acquired them in the first place.

- ▶ Next, track for code quality, code coverage, velocity rates, defect rates, and defect resolution rates (among a billion other things). This helps you figure out just how deep the heap of <bleep> you've gotten yourself into is. Nobody said this was going to be fun.



Now you have enough

- ▶ data to identify the initiatives that will have the most impact, prioritize them, and then make accurate allocations and create realistic budgets and timelines.

With those in hand, develop a simple-to-read tech

- ▶ debt balance sheet that anyone whose job title begins with the letter “C” will recognize. This helps you to simplify the tradeoffs and build your strategy. It helps even more with getting the budget you’re going to need.

Identify the biggest losers.

Not all technical debt is created equal. This is a good thing. Research from McKinsey shows there are typically 10–15 assets that account for the majority of the tech debt in an enterprise.

At the same time, the amount of tech debt between applications can vary by as much as two to three times. For example, one major tech company with more than 50 major legacy apps went hunting for tech debt and discovered that just four apps were driving 50–60 percent of the overall load.

Isn’t it great when priorities set themselves?

Knock it off.

Some orgs benefit by dedicating whole sprints to fixing high-priority technical debt issues. Others set timelines for

solving specific issues and then distribute the work among teams to be done alongside run-of-business work.

Choose your own adventure: Or even mix and match

- i** according to how urgent an issue may be and how much bandwidth your teams (both old / existing and new / acquired) have available to tackle it.

Stop, look, and listen.

Trust your developers when they say it will take a certain amount of time to complete a given migration or remediation properly. *Don't set unrealistic deadlines*, because sacrifices will be made. You're already fixing years of slapdash workarounds and quickie one-off solutions that favored speed over long-term performance — don't go creating brand-new technical debt while fixing the existing.

Like Kyle Simpson said: “There's nothing more permanent than a temporary hack.”

Leave it alone.

More good news: in some cases, the cost of addressing the technical debt of a given asset is simply not worth it. Cross it off your list **after** explaining to your team(s) why it's better to just learn to live with this one. Then maybe buy them some pizza.

HOW TO INDULGE IN FREE SNACKS AND STILL FIT INTO YOUR WORK PANTS



Abundant free treats are one of the great things about being in tech. We work long hours solving hard problems. We build the technology that runs the world. We deserve snacks!

The rise of remote work hasn't altered the nonstop feed-your-face phenomenon one iota. Why would it? When engineers are in the zone with focus work, concepts like “lunch” and “dinner” become but mere abstractions. But snacks? Snacks are always with us, convenient food-to-face fuel that doesn't require moving our steady gaze away from the screen.

Peanut butter-filled pretzels. Chocolate chip cookies. Treats of every flavor, shape, texture, and style. Unless you have iron willpower, chances are you won't be reaching for the carrot sticks, which is how snacks can become too much of a good thing. Your New Year's fitness resolutions never stood a chance and it's starting to look like your waistband won't, either.



If this sounds all too familiar, here are some helpful guidelines for maintaining a healthy snack-life balance.

Don't keep snacks at your desk.

Nope, not even mixed nuts with their high protein and healthy fats. Even when it's a snack that's (kinda) good for you, what's not good is eating the entire package all at once. Which is exactly what will happen if you keep them at your desk. Stand over the sink and eat by the handful like a responsible adult.

Get up and go for a walk.

You're stuck on a hard problem, a stubborn bug, or maybe just waiting for your code to compile, and you think to yourself, *Hmm, maybe those dark chocolate-covered raisins will help move things along.* Trust us, they won't — so move yourself instead.

- i Shake it up:** Get up and walk around. Drink a glass of water. Touch grass. Pet a friendly animal, if available.³ Don't worry: those exquisite morsels will still be waiting as a reward for when you're finished!

Start a snack abstinence competition.

It's really hard to quit a hardcore snacking habit, especially cold turkey. A little friendly competition among coworkers can help the need for victory outweigh the need for Doritos.

- i Challenge a colleague:** See how far into the day you can get without being seduced by a bag of Cheetos or a perfectly salty-sweet Kind bar. Start small: Can you make it to 10 AM? Noon? Notice how much better you feel? Good job! *You've earned a snack.*

Unless you have iron willpower, chances are you won't be reaching for the carrot sticks, which is how snacks can become too much of a good thing.

³Domestic animals, please. Your own, preferably. Consent is required. Do not harass or pursue any wild creatures, like squirrels, trying to sneak a pet; sure, they look fluffy and cute, but nature gave them teeth and claws for a reason.

Make a new rule every day.

Gradual withdrawal from dependency is key to success here. Why not also make it fun with a little gamification?

For example: On Monday you cannot, under any circumstances, eat a chocolate snack. You can have as many pita chips and hummus as you like. But no Hershey's Kisses. Tuesday, no refined carbs — it's dried fruit and assorted nuts all the way down.

And so on. See what you can eliminate as the week goes along. Soon you'll be existing solely on water and sunlight, just like a plant.

Screw it.

Life's too short, the snacks are too good, and the wax on that little BabyBel cheese wheel ain't gonna unwrap itself.

True tales of survival: 20 years of server solitude

I used to work for [REDACTED], a provider of IT production and disaster recovery services. The company was spun from an oil company that had experienced their own disaster. Over time, the company grew to 45 data centers and data recovery centers in five different countries.

At one point, leadership decided they wanted to consolidate some of these different data centers. During the consolidation project, we found an AS/400 server under somebody's desk that had apparently been running for 20 years.

No one had the slightest clue as to why it was there, what it did, or how it had been left to just run on its own for two decades. This actually held up the consolidation initiative because we couldn't unplug it until we knew what it did! Was this server for the company? Was a customer application running on this, or maybe someone's database? *What did this thing even connect to?*

Finally we just had to unplug it and wait to see what would happen.

Right after we unplugged it, some [REDACTED] in Arizona emailed us to say, "Hey, my website stopped working! What did you guys do?" That was it. The only complaint.

To me, the funniest part is that we actually transported this server to the new data center and plugged it again. There was an RTO/RPO agreement with the customer and so we had to make sure that it got plugged back in as soon as we got the email.

That AS/400 is probably still there, chugging along into its third decade, doing its one job.

- [REDACTED], Senior Staff Engineer

CHAPTER



SURVIVING THE WORKPLACE



*“Friends don’t let friends
use us-east-1.”*

– All of Reddit r/SoftwareEngineering

HOW TO SURVIVE A “BLAMELESS POST-MORTEM” WHEN IT’S ACTUALLY YOUR FAULT

So, that thing that happened. You know, The Incident that caused extended disruption for customers. Keyboards were banged, swear words were used, stress balls were squeezed. But everyone pulled together to fix the outage as fast as possible. Now that everything is under control, it’s time for the public-shaming post-mortem.



Luckily for you, your company has adopted Google’s “blameless post-mortem” model! Instead of a criminal investigation, this affirmative approach aims to uncover lessons learned and identify changes to avoid a duplicate future debacle. The core assumption is that everyone makes mistakes from time to time, and the cause isn’t the person but the process.

But what should you do if it really *was* your fault?

Own it.

Even though nobody's allowed to point fingers, everyone knows You Did It. So don't make excuses. With shoulders back and chin up, walk everyone through the timeline of events and explain where things broke down. Spare no details: the first step is to establish a clear and common understanding of What Went Wrong.

Fix it.

Here's where you transform from villain into hero. Because you're not the scapegoat — you're the Error Expert!

- ▶ **First, analyze:** What actions did you take? What effects did you observe? What assumptions did you make? Answering these questions will lead to the real work at hand.
- ▶ **Then, remediation:** Where did the processes break down to allow this incident to happen in the first place? What needs to change? What work can be done to prevent this particular problem from happening again?

Share it.

Write down everything you learned and share the document as widely as possible to make sure disaster never strikes again. (*This* disaster, anyway.)

WHAT TO DO WHEN YOUR RUBBER DUCKY DEBUGGING BUDDY GOES MISSING



Talking out loud to yourself at your desk can be cause for alarm, but for some reason talking to a bath toy is just fine. That's likely because most techies appreciate the value of rubber duck debugging, which explains why cute plastic ducks are perennially popular in onboarding packages and tech conference swag.

But if you just reached for your own adorable plastic debugging pal and thought, *Oh f*ck, where's my duck???*, there may have been fowl play. Now what?

Look for a flock of ducks.

Perhaps your personal debugging ducky has been called upon to assist with a major issue? Sometimes a single duck is not enough and you need to borrow a neighbor's duck to get more ducks on the problem.



A crisis-level fat fingers fiasco may even require summoning every duck in the office. Search your surroundings to see if a Council of Ducks has been convened.

Make a P0 defect ticket in Jira and assign it to everyone on the team.

If you're certain your lucky buddy is the victim of a nefarious ducknapper of

unknown identity, here's a surefire way to get the whole team in on the search.

Task: NCC-1701

Title: Find Mr. Quackers (aka Mark Duckerburg)

Description

As someone who works on this team

I need a certain colorful molded plastic duck returned to my desk

So that we can meet launch date in 3 weeks *with testing completed*

Because Mr. Quackers is an essential tool for me solving blocking bugs w code issues.

Acceptance Criteria

- ▶ Mr. Quackers returns ASAP to his usual place under the left monitor on my desk, unharmed.
- ▶ No questions asked.
- ▶ My job as test engineer on this project is to find bugs in QA instead of customers finding them in production. IT IS NOT A PERSONAL ATTACK ON YOUR CODE.
- ▶ Also, stop stealing my Red Bull from the fridge.

Git blame.

If you *do* know the culprit's identity, this tactic may not help get the duck back immediately — but it will surely help you feel better:

```
$ git blame-someone-else <author> <commit>
```

HOW TO HELP YOUR TEAM BREAK UP WITH BAD TECH

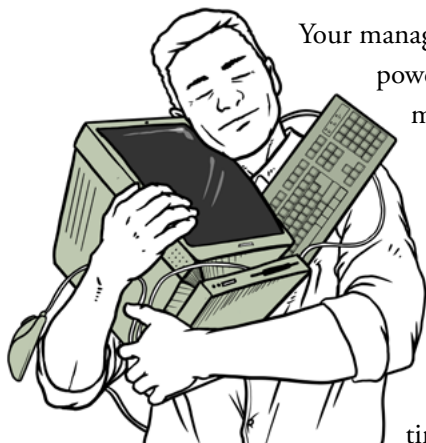


The frustration of being stuck with a terrible tool or slow-moving service that makes our work harder is one of the most common (and least fun) features of a career in tech. But sometimes a solution is so incredibly bad, so maddeningly wrong, that you feel you absolutely must do something. The problem is that everyone else around you seems fine with the status quo and there's pushback when you suggest ways to fix or improve things.

So, how can you roll this boulder on your own?

Get real: is this a needle you can realistically move?

In the immortal words of Kenny Rogers, “You gotta know when to hold ‘em and know when to fold ‘em.” For example: is your VPE firmly wedded to this particular piece of tech dreck that drives you — but *only* you — batsh*t?



Your manager spends some of their own power when fighting decisions made higher up. They'll reserve that political capital for when an issue is truly important to them. A single team member's desire to change to a different technology probably isn't one of those times to gamble.

Document, document, document.

Arm yourself with specific events and examples of when and how this garbage tool broke, slowed down the team, or created risk for the org.

- i Be data-driven:** List the number of weekly hours spent building workarounds because of this POS tech, number of P0 and P1 issues filed, number of poop emojis on the ticket, number of thorns it got in retros...

Do your homework.

No one above the first line of supervision wants to hear why your technology is better. *They want to know what it will do for the business.* You need to show rock-solid business reasons why

your ideas will work in terms of expenditures reduced, work hours saved, customer happiness gained, or downtime averted.

Do the cost-benefit analysis and show your math.

Engage in PsyOps.

It's human nature to resist change, so understanding why your colleagues or higher-ups cling to a terrible / outdated / cruff-stuffed solution is key to winning their hearts and minds.

💡 **Ask questions:** Has it just always been done this way? Does your replacement come with a learning curve? Is it possible they also hold a grudge against the terrible tech but don't want to upset anyone by admitting it?

Uncovering the reasons why this dumpster fire got lit in the first place helps sketch out your argument for extinguishing it.

“Why?” me a river.

You have everything you need! Wrap it all up in a polished presentation or design doc and rehearse ahead of time, because it's likely your sole opportunity to make the case.

Be a leaf on the wind.

In the business world, there's no such thing as a purely technical problem. Sometimes a demonstrably bad solution

is like a zombie that just won't die, powered by unseen forces such as politics, the current tech stack, other systems it has to integrate / interface with, or juicy blackmail that the solution's sales rep has on your boss's boss's boss — things that are far, far beyond your control.

If you come from wanting to build a great product and you have data and documentation to back up your thinking, chances are people will at least hear you out. If they still insist on the original spec, well, you did all you could. A possible silver lining is that your diligence and entrepreneurial spirit get you noticed, opening up new opportunities...ideally on a different team where you'll never have to touch that POS again.

HOW TO LURE YOUR TEAM BACK INTO THE OFFICE WITHOUT INCITING A MUTINY

Back in March 2020, when we all said “See you in a couple weeks!” and walked out of the office, we had no idea those first few weeks of remembering to unmute while sitting next to a pile of laundry would turn into months. Then years. Now that it’s reasonably safe for all of us to once again share the same air, however, many companies want their people back on-premises most, if not all, of the time. Heck, even Zoom asked their employees to return to the office.



At the same time, many employees haven't been too keen to embrace return to office (RTO) plans and resume their commute or relinquish the joy of working in their jammies. They need a more compelling reason than <corporate voice> *Because we said so*. The fact is, people have come to expect flexibility and autonomy around how, when, and where they work.

How can you persuade your WFH employees to voluntarily get on board the SS RTO?

Accept that hybrid is now reality.

The percentage of full-time employee butts in desk chairs right now is the same percentage that willingly came back to the office. The remaining percentage (probably a big one) is the one you need to reach with an alternative policy (i.e., more flexible than the 9-to-5 weekday schedule that was the prepandemic standard for most offices).

- 💡 **Embrace the split:** Craft a part-time-in-office / part-time-remote policy that works for both sides. Otherwise, plan on backfilling roles for the 39 percent of workers who say they'd leave their current job if they were mandated to resume a full-time office presence.

If remote employees grumble whenever RTO plans are mentioned, ask why they resist changing into hard pants and coming to the office.

Listen up.

If remote employees grumble whenever RTO plans are mentioned, ask *why* they resist changing into hard pants and coming to the office. For example, 42 percent of parents with children under 18 say that remote

work allows more flexibility with school, daycare, and other family responsibilities. (This overlapped almost entirely with the number of parents who reported using “PAWpatrol99” as their Okta password.)

Overall, actually *getting* to the office is a widely loathed aspect of RTO (59 percent), along with dressing more formally (43 percent). Recognizing these issues lets you craft an enticing RTO that addresses employee needs with benefits like hybrid scheduling, commuter perks like paid parking or transportation, or a more relaxed dress code.

Go both ways.

What does the business want from having employees in office: in-person collaboration? Higher productivity? Better client interactions? Evidence that your employees do actually exist

from the neck down? It's possible to achieve these outcomes while still offering your workers some flex in their schedule.

Successful companies with productive hybrid cultures do one or both of two things:

1. Create a policy of targeted days for all employees to be onsite to maximize cross-org together time. Tuesdays, Wednesdays, and fantasy football draft days are peak for most hybrid offices.
2. Require a set number of in-office days per pay period / quarter / month, but leave it up to the individual to decide which ones.

Keep it equitable.

Speaking of going both ways: A BambooHR survey found that 10 percent of independent contributors reported that executives at their companies had the option to work from home, but no one else did. Extend the same trust to your employees that you do to your senior personnel.

Reopen the treat shop.

Between the pandemic and economic downturn, many companies cut expenses by eliminating in-office benefits like gourmet coffee machines, smorgasbords of exotic free drinks and snacks, and catered office lunches.

i **Keep it simple:** You don't need to bring back over-the-top perks like free laundry service or beanbag rooms or five kinds of kombucha on tap — just the standard techie treat trinity of on-demand caffeine, sugar, and carbs. After all, right now your WFH employees have to make their own coffee and buy their own snacks. Just saying.

Bring the hammer down.

When all else fails, if you absolutely, positively need your people present onsite and can't convince or cajole them into returning, there remains one option: threaten them (AKA “pulling a Google”). Alphabet Inc. initially asked workers to be in the office at least three days each week. Then they announced that performance reviews can take consistent office absences into account, and badge records would be used to identify any Googlers stubbornly refusing to change out of their sweatpants and report in person.

HOW TO SURVIVE A MEETING THAT SHOULD HAVE BEEN AN EMAIL



Some meetings can feel like the equivalent of ransomware, only in a conference room. Any time a particularly dreaded session looms on your calendar and it's not feasible to fake an emergency event (network failure, heart attack, etc.) to get out of going, these strategies can help minimize the physical and psychological trauma.

Keep those eyes on the prize.

This is crucial: you need to at least appear like you're paying attention. For virtual sessions, simply look up every now and then, directly at screen center; if it's a live speaker, brief eye contact should do the trick. Nod occasionally, as though you're agreeing with the latest point. But not *too* enthusiastically, or you might end up in charge of whatever it is they've been droning on about for 45 minutes.

Memorize this phrase.

If you're abruptly summoned back from pleasant daydreams of being anywhere but in this meeting by someone asking for your thoughts on the current ideation session, you need to be prepared. These words will save you every time: "I'll be happy to circle back and synergize my learnings with everybody later, but right now I'm drinking from the firehose."

This clearly establishes you as a deep thinker who requires time to absorb all this wisdom, and "later" never arrives *because this meeting will never actually end.*

Play Lingo Bingo.

Like any endurance event, surviving this marathon meeting / strategy session / brainstorm requires proper hydration. There's nothing like a little drinking game to stay hydrated.



For virtual sessions, simply look up every now and then, directly at screen center; if it's a live speaker, brief eye contact should do the trick.

Lingo Bingo⁴ is the perfect pastime: pick a phrase or three that makes you die a little inside every time you hear it, then swig from your beverage whenever someone utters it. Research shows a 1:3 ratio of caffeinated to noncaffeinated beverages is optimal. (The numerous bathroom trips this game necessitates are a feature, not

a bug.)

Take “notes.”

Taking notes during a meeting can help you stay focused and retain important information. It can also help you retain your sanity when those “notes” include intricate doodles, spur-of-the-moment caricatures of Daryl, who always wants to *just go back a few slides*, or perhaps scorekeeping for the Lingo Bingo championship.

⁴Experts caution against using actual Bingo cards, even with like-minded coworkers, because: (1) it's difficult to disguise the cards; and (2) it's poor form to shout *BINGO* during Jerry the ABM's presentation on disintermediating mission-critical metrics.

Ransack the snacks.

Any decent extended meeting or brainstorm session features a buffet of goodies, so make like a raccoon and snag a pile of your favorites. Be shameless. After all, this is a survival situation.

💡 **Make a game of it:** How silently can you tear open each package of trail mix? (Or how loudly, if you'd prefer to tip your meeting participation level from passive-aggressive to aggressive-aggressive.)

Purge your inbox.

Keep yourself busy accomplishing something useful while making it look like you're attentively keeping notes. No better time than a marathon meeting to get cracking on those 3,578 unread. Definitely have a backup strategy in mind, though, in case you achieve Inbox Zero before the meeting ends.

True tales of survival: Surviving a hurricane with a bucket brigade

It was October 30, 2012. I was working for [REDACTED] Software at [REDACTED] in Lower Manhattan. Squarespace and Peer 1 Hosting were in the same building, and we shared an on-premises data center. Unfortunately, this was our only data center at the time, because we had just decommissioned our DR location in an effort to move everything to the cloud. We had tested our code and infrastructure services on AWS, but we had yet to upload our vast amounts of data. And then Superstorm Sandy hit.

The building had backup generators. These were in the basement, along with the tanks holding diesel to fuel these generators. When the storm hit, though, lower Manhattan flooded and this basement was completely filled with water – one floor above, in the lobby, the water was four feet deep. The generators could not go online due to being under water, but we were lucky in that the data center had its own small backup generator with its own fuel tank. Less lucky was that it was at the top of the building on the 17th floor.

When the storm passed, the data center was still online. The folks operating the data center were there 24 / 7, sleeping on the floor, and they were communicating with all of us about how much fuel was left. Everybody was trying to lower usage in the data center to preserve fuel and extend the running time, but the generator was running low on fuel so it was only a matter of time. But then a miracle happened.

It so happened this was a colocation data center, and someone from one of the other [REDACTED] customers there was somehow able to secure a load of diesel fuel – put it on their personal Amex for \$25,000 – and the flooding had gone down enough by then that the truck could actually make it to our building. They were also able to round up some empty 55-gallon drums.

I came in to participate in keeping the data center alive, and people from other companies in the building came, too. The truck showed up and unloaded diesel into the drums. From there we emptied it into 5-gallon buckets and carried it up 18 flights of

stairs to the data center and poured it into the fuel tank of the generator. Then we'd go back down and do it again.

We had shifts of people doing this for almost 24 hours. A gallon of diesel fuel weighs about eight pounds, so you were carrying two buckets each weighing 30 to 40 pounds. We didn't have any lids for them, so the fuel slopped out and the stairwell got very slippery and filled with diesel fumes. We had lights barely working in the data center but in the main part of the building it was zombie town, apocalypse pitch dark, the only light was your headlamp or flashlight.

After my shift was over I managed to get a cab home to the Upper East Side. I was covered in diesel fuel, feeling incredibly sick from the fumes, and I couldn't wait to get a shower. My wife's like, *We are not even going to try washing those clothes*, so we double-bagged them and threw them in the trash.

Simultaneous to staffing the bucket brigade that was keeping the data center alive, our [REDACTED] team also had our last-ditch emergency backup strategy going, which was to copy all the data to external USB drives and take those drives to an Amazon data center in Virginia. It was vast amounts of data, which you can't upload quickly over a typical internet connection, and they would only accept delivery by courier so we had no choice but to take it to them. We had to figure out, *who's got a car, how much gas have you got, how far can you go?* Sandy had hit the whole East Coast and many gas stations were closed or sold out.

The story has a happy ending. The data was on its way to Virginia, they got the basement pumped out and everything back on and in working order in the building. The data center stayed alive through one of the most destructive storms in history. And me? I learned a big lesson about having an active disaster recovery plan at all times. What's gonna happen if the worst-case scenario goes down? What are you going to be doing - are you ready to take that risk? Because you never know when the next super storm is coming.

- [REDACTED], [REDACTED] Cloud Engineer

CHAPTER



SURVIVING THE FUTURE



“If you think good architecture is expensive, try bad architecture.”

– Brian Foote and Joseph Yoder

HOW TO SURVIVE A KAIJU ATTACK ON YOUR DATA CENTER



As anyone who lives near the Pacific Ocean can attest, the threat of sea-born interdimensional creatures wreaking havoc on infrastructure is an ever-present one. Once Godzilla and friends break land, it can often take *hours* for a team of giant mechs to deal with them. In the meantime, you can say *sayonara* to bridges, skyscrapers, and your data center.

Any cloud region name containing “West” or “Japan” or “Korea” puts you at elevated risk of Kaiju-related data destruction. But the truth is, unexpected catastrophic threats to your data can (and will) emerge any time, anywhere.

Here’s how you can avoid annihilation.

Create a survivability plan.

As Amazon CTO Werner Vogels warns, “Everything fails all the time.” You need a strategy to prepare for the inevitable demise of your cloud provider’s data center, whether it

crumbles from the force of a 15,000-ton Kaiju or a 1.5-pound squirrel.⁵

Start by determining the optimal size and survival characteristics for your app's data cluster. A little fine-tuning will help you determine what category of Kaiju your cluster is prepared to withstand.



Set custom availability goals at the database, table, and row levels.

No matter the nature of the disaster — be it an airborne Mothra strike or sharks gnawing undersea cables⁶ — your critical data will be available.

Replicate, replicate, replicate.

Database replication (3x is the ideal starting point) is a survival must for single-region apps. If a replica is lost, your remaining replicas continue serving reads and writes.

⁵Google “squirrel outage” and you’ll find a lengthy record of both recent and historic incidents of squirrels causing data center power outages.

⁶Also an astonishingly common problem.

Determining how many replicas you need depends on a number of factors, including customer locations, workload criticality, and risk of Maximum Mega Kaiju-category events.

Disaster-proof your app.

A multi-region architecture ensures that, even when your cloud provider's services go dark in one availability zone — whether due to an epic battle between Rodan and Godzilla to determine the new King of the Monsters, or a backhoe severing a data center network cable — your app lives on.

Alert the Mecha Kaiju Response Team.

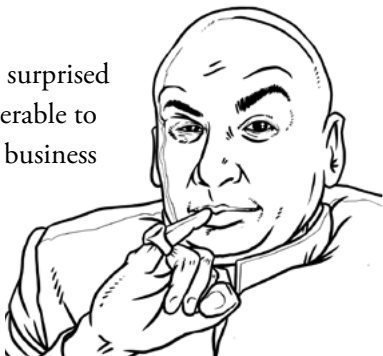
They're probably already aware of the situation, but it's good to file a ticket anyway.



HOW TO SAVE YOUR COMPANY \$1 BILLION (NO, REALLY)

Government efforts to control how data produced within their borders can (and cannot) move around the planet are increasing faster than your AWS egress charges. Data privacy laws like the EU's General Data Protection Regulation (GDPR) are pushing companies to store user data within the country where it's collected — with potentially stratospheric fines levied against companies who don't comply. Meta learned this the hard way after being socked with a €1.2 billion (\$1.3 billion) fine for violating GDPR by moving European user data to US data centers.

Many US-based enterprises are surprised to learn that they, too, are vulnerable to GDPR fines even though their business doesn't target European customers. (Sorry, we don't make the rules.) In addition, many other countries



outside the EU as well as US states <side eyes at California> are passing their own privacy laws. These regulations are often complex, but fortunately there are some simple steps most companies can take to meet basic data privacy requirements and avoid fines. Especially the ones carrying ten zeroes.

Be like Cookie Monster.

If your website or app tracks user behavior (if it doesn't, are you even marketing?), then placating data protection policies requires gaining user consent.

To protect visitors' privacy, sites should have a popup that allows visitors to accept or decline consent for third-party cookies upon their first visit. This popup should also include a link to your privacy policy.

Cover your privacies.

We know, we know: who actually reads user privacy statements? Regardless, don't just ask ChatGPT to whip one up for you. The Venn diagram of the people who DO read privacy policies and the people who can fine your business for violating data privacy regs has an approximately 99.999 percent overlap.

- i Be aware of relevant privacy rules:** You must explain how your app or site collects user data and how it's used. For example, your privacy policy should specify how a

customer's contact information will be used (marketing lists, third-party partners, feeding your new AI) and how contact will be made (email, phone, standing in their front yard holding up a boombox), as well as how customers can opt out of these lists and communications.

Get cryptic.

Many privacy laws, including GDPR, require businesses to secure and protect user data through encryption. This ensures that, in the event of a breach, the stolen data is useless. (Take that, hackers!) The upside is doing this also reduces your risk exposure if your company gets hacked, since encryption deters both Russians and robots equally.

Keep (user) options open.

GDPR and similar laws require that businesses give users access to their information upon request. You must provide a way for users to request their saved data. Users must also have a way to withdraw consent and have their data deleted.

- i Make it good:** If your version of this takes too long or otherwise frustrates users, they can report you, leading to one or more regulatory agencies poking around in your sensitive compliance areas (ouch).

Keep it strictly personal.

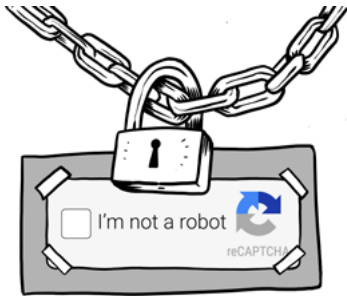
GDPR and its regulatory ilk govern every type of user information your business can conceivably collect. Literally every nanobyte of data you can hold about customers must be compliant. This includes: email addresses, device information, user behavior, your dog's favorite baseball team — it's all fair game.

Leave your data where it belongs.

Reduce risk and make your life easier by choosing tools and services purpose-built to address compliance. Use CRM tools that automate data collection and management according to GDPR guidelines. Choose a database that allows full control over encryption and who has access to your data, with built-in data localization for row-level control over where user data resides.

Such tools can't completely solve the compliance problem for you, but they do hand you the (encryption) keys to solve it yourself, according to your specific business needs and which government agencies might be peering over your shoulder.

HOW TO PROTECT YOUR ASS(ETS) FROM AI ATTACKS



Until now, companies mainly had to worry about opportunistic hackers hunting for open servers to exploit, the equivalent of walking down a street trying all the car doors in the hopes that one's been left unlocked. Now, thanks to AI, hackers get right to the good stuff by targeting specific servers and using chaining language models and other AI capabilities to perform deeper probes into any internet-exposed infrastructure — like port scans, only on steroids. AI also will enable them to exploit vulnerabilities more rapidly and with greater sophistication than ever before.

But there are analog methods for foiling even the cleverest of AI-driven attacks that are simple, straightforward, and don't even require AI.

Patch all known vulnerabilities.

We shouldn't even have to say this, but: 60 percent (!) of all data breaches are caused by companies failing to patch and update software in a timely fashion. Don't be them.

Go on a security tool scavenger hunt.

The average large enterprise has 70+ tools (!!) helping them with their security posture. This scenario creates more opportunities for missing subtle-but-important signals than star-crossed couples in a '90s rom-com.

- i Centralize everything:** Audit all your security tools and keep only the essential ones. Next, implement one master data management (MDM) tool to provide a single source of truth for the organization's essential business data.

Create an allowlist.

Many companies no longer have a clear understanding of their own technology portfolios. Gen Z and Millennial employees in particular don't think twice about using browser-based productivity tools and downloading new applications with click-through agreements.

Employees download / grant permissions to apps without alerting IT or the security team, and with scant

understanding of potential data leak implications. The result is an attack surface the size of Jupiter.

- i Set a standard:** First, create an org-wide standardized “allowlist” of approved tools and services. Next, implement a process that ensures any new additions pass through security review before they go on anybody’s laptop.

Go visit granny.

Just about every enterprise org has “golden girls” — applications that roll along with no new feature development, dependably doing their job year after year. These long-lived workhorses often go untended, possibly because the original developers are no longer around and everyone else is terrified to touch the codebase.

Such venerable applications likely contain vulnerable code or dependencies (see also: log4j). It’s time to train new developers on these software beasts of burden to update them and build familiarity with the codebase.

60 percent of all data breaches are caused by companies failing to patch and update software in a timely fashion. Don’t be them.

- ❗ **Cut the cord:** If for any reason an application is too brittle to change, a drastic yet effective second option is to leave it untouched but remove inbound internet access.

Shift left, but not too much.

The core of DevOps is development teams taking full responsibility for their own services or applications, from build to test to deploy. But there is such a thing as too much responsibility. Expecting the average engineer to be fluent in cloud infrastructure, databases, network config, pipeline tooling, caching, Kubernetes, and security is a tad bit unrealistic.

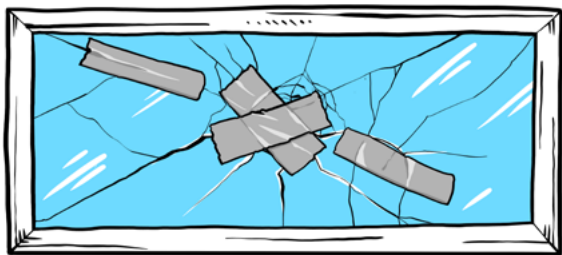
Realistically, how many teams have both the experience and skill sets to adequately handle all security concerns across their application stack, cloud environment, and toolchain?

Security is increasingly a job for a focused expert, not a frazzled dev team on a tight launch deadline. Especially now that the bad guys have AI, too.

HOW TO DUCT TAPE A SHATTERED SINGLE PANE OF GLASS

In the beginning there was on-premises, with key punches, batch jobs, and a single OS. Then the people cried out, “Give us cattle, not pets!” And so the great ones bestowed upon the world virtual machines and hypervisors. Chief among these gifts was VMware, a single pane of glass for managing all enterprise workloads. Peace lay upon the (bare metal) land.

But, lo, ages passed and paradigms shifted. Enterprises moved into the cloud, which visited a terrible plague of complexity upon operating infrastructure. It was a time of great chaos: divisions rushed to create their own cloud provider rather than



standardizing on one, such that the same data would often get uploaded to the same cloud multiple times. In desperation, some divisions even rendered unto their vendors more gold to do the support and monitoring *of their own workloads*. All across the land, the people mourned the shattering of their single pane of glass.

It falleth upon technical leadership to gather together all those pieces and restore that single pane of glass. We hold up to you these four commandments to guide your path.

Consolidate.

Thou shalt reduce complexity.

Bring databases together. Heck, bring data centers together. Standardize on integration tools and endpoint controls. Centralize security. Implement a single master data management (MDM) tool. Use software / services suites as much as possible.

The goal is to pare down the number of strategic infrastructure vendors to five (best) or seven (if thou must).

Standardize on Kubernetes.

Thou shalt have but one orchestrator.

Kubernetes won the container wars, and it's ubiquitous. That makes it the only technology with the potential for returning

again to a single control pane. Most of the software world has already moved or is moving to K8s anyway. It's critical that all new SaaS and PaaS workloads run Kubernetes.

Over time, you can work backwards to migrate existing workloads as well. Make K8s your single pane of glass to run enterprise workloads — no matter where they run.

Welcome automation and AI.

Thou shalt require highly automated solutions.

Visibility is crucial, but it can only do so much. Humans will never be as fast as computers, so routine operations need to be operated by software, not people.

For important tasks that have traditionally remained stubbornly manual, such as SQL query tuning, look for ways to apply AI-driven tools.

Preach it.

Thou shalt deliver the word unto thy people.

As you consolidate and standardize, teams are going to fight to keep their favorite tools and comfortable workflows. There's no getting around it: some people are going to be seriously pissed.

But operating with multiple dashboards adds complexity and kills velocity. Emphasize and evangelize the benefits that everybody will enjoy from consolidation, standardization, and automation, like more time to experiment with new ideas and less — or even zero — pager duty.

Eventually everyone will be right there with you, gazing upon your shining, newly intact, single pane of glass.

<Cue heavenly music>

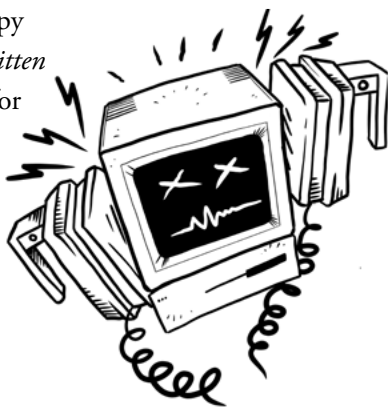
Visibility is crucial, but it can only do so much. Humans will never be as fast as computers, so routine operations need to be operated by software, not people.

HOW TO BUILD AN EMERGENCY LIFE SUPPORT SYSTEM FOR A LEGACY APP

Migrating existing apps to the cloud is never straightforward, especially when you're refactoring a legacy monolith (insert COBOL jokes here). But here's the important thing to understand about venerable apps in long-established companies: *these are the apps that make the moolah. Do not, under any circumstances, break them.*

But now your CIO wants everything to go to the cloud. *Everything.* There's just one wee problem: your org's main honeypot app (written in the floppy disk era) must be completely *rewritten* — but there's no time or budget for that in the migration plan.

So! How do you go about moving critical legacy tech to the cloud without breaking anything?



Build a [hybrid] bridge and get over it.

Hybrid cloud is a combination of two (or more) computing environments that share information with one another and run a uniform series of apps. In this case, we're almost certainly talking about one bare metal or virtual environment, where your legacy stuff was born and has lived all its life, connecting to a public cloud.

Link your co-lo data center to your public cloud platform.

We're talking about a direct, dedicated connection using a service to link your cloud and on-premises networks. This lets you span environments without compromising performance and ensures smooth and reliable data transfers — even at massive scale.

Build APIs in front of your legacy app(s).

Use your favorite software development paradigm. Keep it simple. Then...

- ▶ **EITHER:** Strangle that app.
Keep these APIs healthy and rely on overlay networking until you finish strangling that app (or apps). Then, once there's finally budget and resources, refactor them and move them completely over to the cloud.


- ▶ **OR:** Make the bridge permanent.

Seriously: stay hybrid. If you have a functioning and stable legacy app that rarely changes but can't be retired easily, it makes sense to keep it intact instead of trying to refactor for the cloud.

That way, no one ever has to be the one blamed for breaking *The Sacred Code That Is So Important We Haven't Touched It In Decades*.

This is a surprisingly common pattern. For example, financial services orgs often have a critical need for scalable transaction processing while still operating from an on-premises mainframe database. In such cases, this build-a-bridge hybrid cloud model makes a lot of sense and still allows for leveraging services like AWS Lambda for true pay-for-use computing needs. It can also lend OG street cred now that repatriation is a thing. *Move data back to bare metal? We never left!*

HOW TO JUSTIFY BLOWING UP YOUR TECH STACK WHILE PLAYING A ROUND OF GOLF



Face time with executives, board members, or anyone else directly overseeing your IT budget is precious. You'll take it whenever and wherever you can get it, even if the "when" is 7 AM tomorrow and the "where" is on the golf course. After all, the secret to success in golf is to turn three shots into two, and the same applies to persuading leadership that a major refactor is needed. A lot is riding on your ability to explain the benefits of blowing up your current tech stack while not embarrassing yourself on the green.

Here's how to survive the minefield of balancing business with the sport of kings during your cart and fairway time.

Understand the rules.

Golf has some pretty obscure rules. For example, the "Loose Impediments in the Hazard" rule lets you pick up foreign

objects while simultaneously treating the ground like it's lava. Nimble removing a twig without touching the bunker will not go unnoticed by your colleague. Once they recognize your knowledge of and adherence to the complex rules of golf, they're ready to appreciate your deeply technical GDPR data sovereignty strategy.

Tee up the conversation.

Sure, your (very) senior colleague invited you for this round of golf at Monolith Hills to *eventually* talk business. But knowing when and how to bring up work requires just as much skill as a long par 5 with a narrow fairway and water hazard. When the moment feels right, here are a few golf-related segues to try:

“Dang, you’re locked in today! Which is a good thing when you’re playing golf. Vendor lock-in, on the other hand, is a real problem.”

“I don’t think I can hit the green from here. Yup, I have a real geographic reach problem. It’s simply impossible for me to deliver this ball to where it needs to go without a lot of latency.”

“Ouch, can I take a mulligan? Of course, there are no mulligans when it comes to loss of customer loyalty and lower CSAT scores that result from single-cloud outages.”

Carry only a single club.

After a few holes spent watching you putt with a 7-iron, your colleague should truly get it: no single provider can accommodate every possible business need.

Quote *Caddyshack* liberally.

No one knows why this is still a tradition — just roll with it. *The shortest distance between two points is a straight line in the complete and opposite direction.*



About the Author

Michelle Gienow is a recovering journalist turned software developer. In her spare time, she also teaches wilderness survival skills like tracking, firemaking, and which plants you can eat without dying. Writing a book about surviving disasters in the tech world was an unexpected but enjoyable overlap of these skill sets.

She is also co-author of the O'Reilly book, *Cloud Native Transformation: Practical Patterns for Innovation*.

About the Illustrator

Giovanni Cruz first realized he could draw when he was bitten by a radioactive pencil. To the amazement of his family, this has become an actual paying career. He can be often be heard mumbling, *“with a great shaperner, comes a good point.”*

About Cockroach Labs

Cockroach Labs is the creator of CockroachDB, the most highly evolved, cloud native, distributed SQL database on the planet. Helping companies of all sizes — and the apps they develop — to scale fast, survive disaster, and thrive everywhere.



LEARN TO BUILD WHAT CAN'T BE KILLED



The essential reference guide to CockroachDB, the world's most evolved distributed SQL database, shows how to architect apps for effortless scale, ironclad resilience, and low-latency performance for users anywhere.

For a free copy, scan this QR code or visit:
cockroach.ch/cockroach-book

“We have literally been unable to kill this thing.
No matter what we've thrown at it.”

– Cloud Platform Architect, Bose



Sometimes keeping applications online and healthy is the least of your problems. Unforeseen, and occasionally bizarre, disasters (fire, flood, fat-fingered YAML files) can strike your workloads or work life at any time.

What will you do?

This illustrated guide offers advice for surviving a variety of scenarios, such as:

- ✔ How to protect your org and your data from AI-powered attacks
- ✔ How to survive a delusional delivery date
- ✔ How to get WFH employees to RTO without a mutiny
- ✔ How to build an emergency life support system for legacy apps
- ✔ How to survive a Kaiju attack on your data center

Read this guide for tips to survive common disasters, bizarre mishaps, and extreme situations that may threaten your workloads, services, and sleep — plus whatever is left of your sanity.