Perma-web

Neil Kandalgaonkar

2024-06-27 · Vancouver, Canada

Our Networks 2024

A few years ago,
I redid my personal website

August 12th, 2012

ToorCamp 2012

I attended ToorCamp 2012, the "five day, open air, tech camping event" in Washington State.


People's Republic Of Nosebridge by Neil

We managed to find the one place in Washington state where it is freezing cold and damp in the summertime. I think it just touches the current that eventually makes the Bay Area so frigid.

Things I would never do anywhere else: program a robot to dance, hold a sword receiving blasts of purple lighting bolts from a Gauss gun, worship at the Church of Robotron, get a preview of a cool interactive toy Star was working on, help assemble a tower built of pipes to validate one of Miloh's theories about constructing large structures with struts of uniform length, play foosball on a table that automatically keeps score and lights up when points are scored, attend a bicycle jousting event....


A man and his lightning gun... by tipsen

Incidentally, it is fun to terrorize small children with robots. Hacked on MC Hawking's software with some other NoiseBridge folks so it could be remotely controlled. Was a bit surprised that it was all in Java (it's some standard package for doing robots, apparently). That made consuming a line-oriented protcol super annoying, and my more unixy style of programming is a bit foreign to Java, but we eventually made it work.

Conversations I would never have anywhere else: transgender hacker still adapting to her new life, Egyptian activist on exactly how influential the internet was in the 2010-2011 revolution, how to set up your own GSM phone service

> – *Why isn't the phone network working yet ?*
>
> – *[complex explanation about crystals and synchronization]. This will take another six hours.*
>
> – *Isn't there some quick and dirty way to do this?*
>
> – *This IS the quick and dirty way.*

Spontaneously taught: Haskell, how unix process forking works

Spontaneously learned: lock picking, the chemistry of baking, how to turn kelp into a musical instrument, Bitcoin, blinding, LASERS

It's the kind of conference where you overhear a woman talking about her sex toy experiments and how nobody else seems to do this, and not one but two bystanders inform her she should be talking to qDot.

I helped make that thing you like on the Internet. Google, Upcoming, Flickr, and Wikipedia. More info...

Recent Posts

Answer to Ed Felten's Hello World puzzle

Manliness for manly men like me

HeartForth

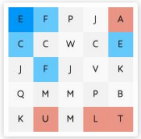
Thoughts about Outernet

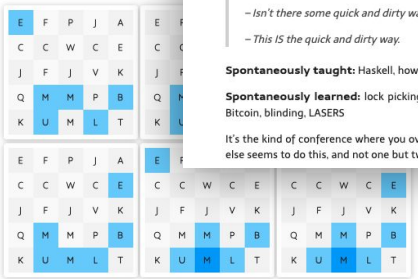
Less represented and loving it

Tweets by @flipzagging


halowolf test by Neil


casey hammer by Neil





This gives us a list of six *moves* we could play on the board, represented in bitmap form, so it's just a list of six integers:

Hacker-friendly choices

Octopress - static generator

- Jekyll
  - Markdown/YAML
- Ruby
- CSS, SCSS, LESS
- Octopress extensions
  - Pygments; syntax coloring
  - Octopress-Flickr; hosted images
    - Fancybox.js
- Official Twitter Javascript API

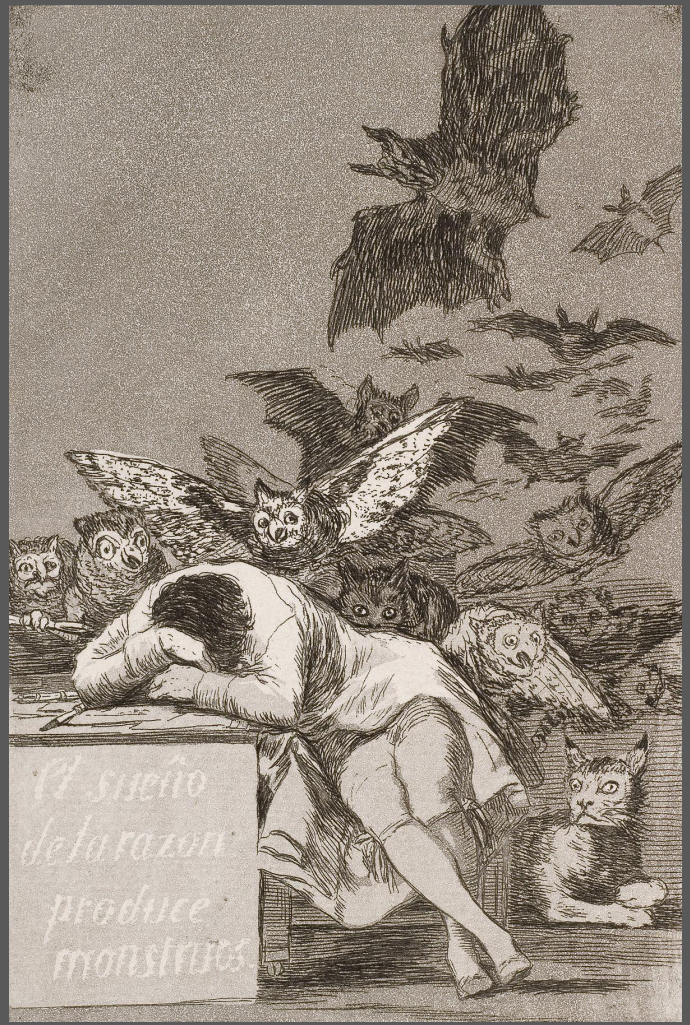
In just a few years,
none of it worked

Hacker-friendly choices?

Octopress - static generator

- Jekyll
  - Markdown/YAML
- Ruby
- CSS, SCSS, LESS
- Octopress extensions
  - Pygments; syntax coloring
  - Octopress-Flickr; hosted images
    - Fancybox.js
- Official Twitter Javascript API

Hacker-friendly choices?



What went wrong?

Everything will break

...and sooner than you expect

“Permacomputing” (from permacomputing.net)

Have few dependencies

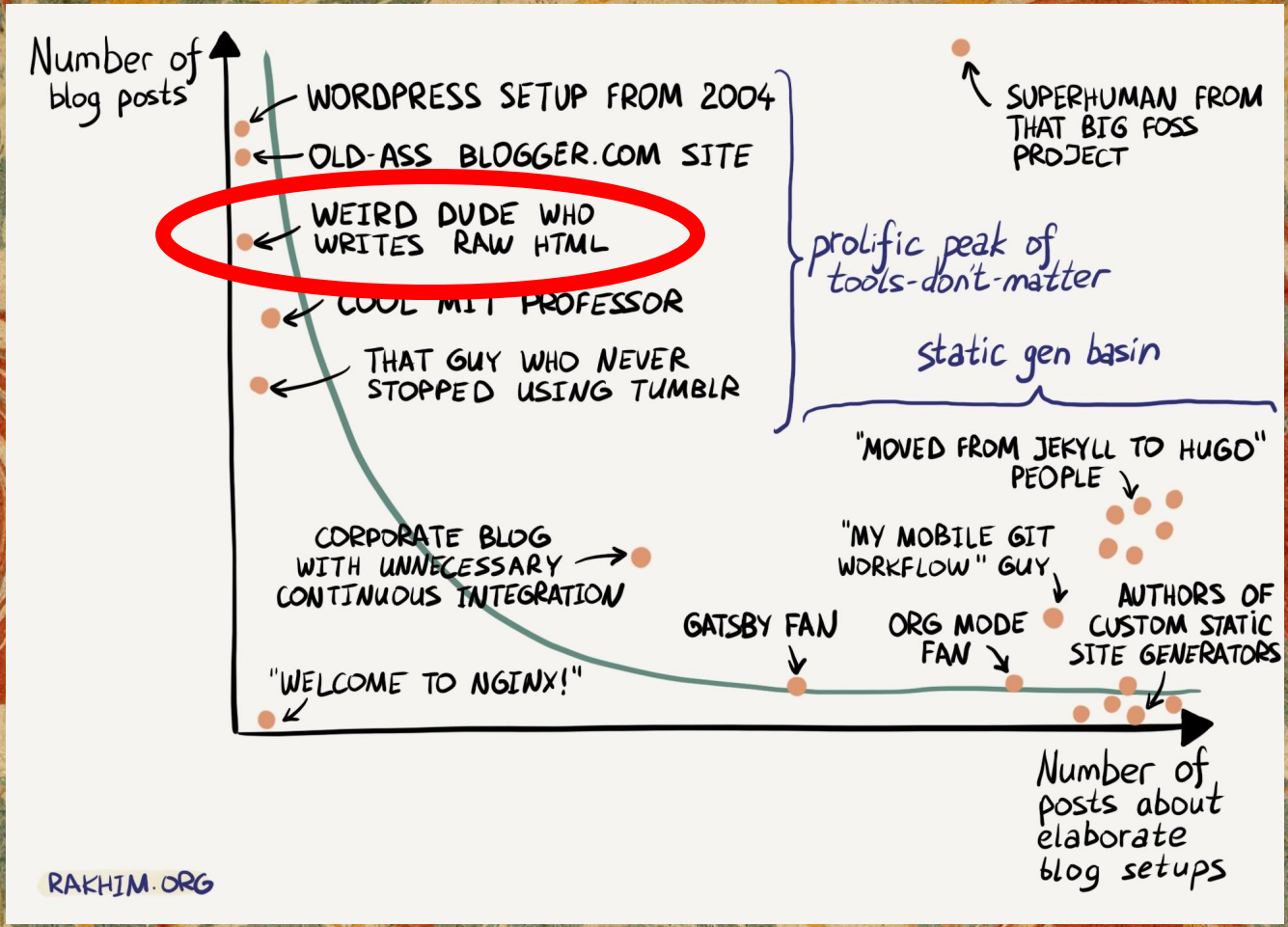
Avoid pseudosimplicity

Accumulate wisdom and experience rather than codebase

Human-scale

Hope for the best; prepare for the worst

Design for descent (degrade gracefully)



https://rakhim.org/honestly-undefined/19/

The container throttling problem | Patreon

---

*This is an excerpt from an internal document **David Mackey** and I co-authored in April 2019. The document is excerpted since much of the original doc was about comparing possible approaches to increasing efficency at Twitter, which is mostly information that's meaningless outside of Twitter without a large amount of additional explanation/context.*

At Twitter, most CPU bound services start falling over at around 50% reserved container CPU utilization and almost all services start falling over at not much more CPU utilization even though CPU bound services should, theoretically, be able to get higher CPU utilizations. Because load isn't, in general, evenly balanced across shards and the shard-level degradation in performance is so severe when we exceed 50% CPU utilization, this makes the practical limit much lower than 50% even during peak load events.

This document will describe potential solutions to this problem. We'll start with describing why we should expect this problem given how services are configured and how the Linux scheduler we're using works. We'll then look into case studies on how we can fix this with config tuning for specific services, which can result in a 1.5x to 2x increase in capacity, which can translate into $[redacted]M/yr to $[redacted]M/yr in savings for large services. While this is worth doing and we might get back $[redacted]M/yr to $[redacted]M/yr in TCO by doing this for large services, manually fixing services one at a time isn't really scalable, so we'll also look at how we can make changes that can recapture some of the value for most services.

The problem, in theory

Almost all services at Twitter run on Linux with the CFS scheduler, using CFS bandwidth control quota for isolation, with default parameters. The intention is to allow different services to be colocated on the same boxes without having one service's runaway CPU usage impact other services and to prevent services on empty boxes from taking all of the CPU on the box, resulting in unpredictable performance, which service owners found difficult to reason about before we enabled quotas. The quota mechanism limits the amortized CPU usage of each container, but it doesn't limit how many cores the job can use at any given moment. Instead, if a job "wants to" use more than that many cores over a quota timeslice, it will use more cores than its quota for a short period of time and then get throttled, i.e., basically get put to sleep, in order to keep its amortized core usage below the quota, which is disastrous for tail latency[1].

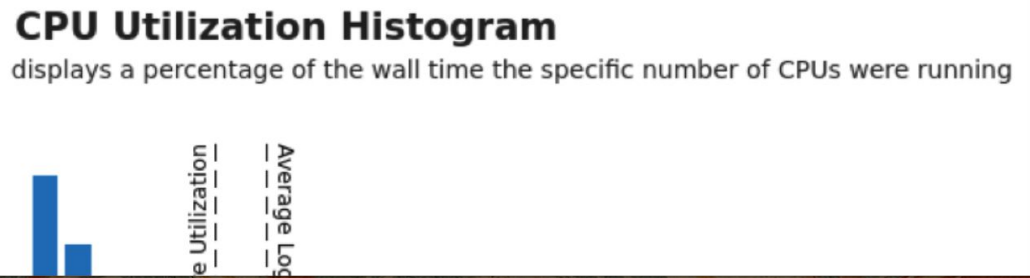
Since the vast majority of services at Twitter use thread pools that are much larger than their mesos core reservation, when jobs have heavy load, they end up requesting and then using more cores than their reservation and then throttling. This causes services that are provisioned based on load test numbers or observed latency under load to over provision CPU to avoid violating their SLOs. They either have to ask for more CPUs per shard than they actually need or they have to increase the number of shards they use.

An old example of this problem was the JVM Garbage Collector. Prior to work on the JVM to make the JVM container aware, each JVM would default the GC parallel thread pool size to the number of cores on the machine. During a GC, all these GC threads would run simultaneously, exhausting the cpu quota rapidly causing throttling. The resulting effect would be that a subsecond stop-the-world GC pause could take many seconds of wallclock time to complete. While the GC issue has been fixed, the issue still exists at the application level for virtually all services that run on mesos.

The problem, in practice [case study]

As a case study, let's look at `service-1`, the largest and most expensive service at Twitter.

Below is the CPU utilization histogram for this service just as it starts failing its load test, i.e., when it's just above the peak load the service can handle before it violates its SLO. The x-axis is the number of CPUs used at a given point in time and the y-axis is (relative) time spent at that utilization. The service is provisioned for 20 cores and we can see that the utilization is mostly significantly under that, even when running at nearly peak possible load:



Can a “permacomputing” site look nice?

Write simple, raw HTML

…automatically transform it to *better* HTML

…in stages

…if any transformation stage breaks, the original HTML is passed to the next phase

Failure embracing HTML transformation pipeline

Simple source HTML

↓

Add header boilerplate

↓

Add navigation boilerplate

↓

Add syntax coloring

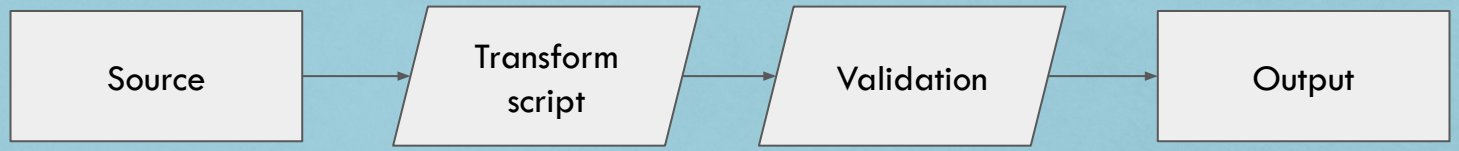
↓

Add footer boilerplate

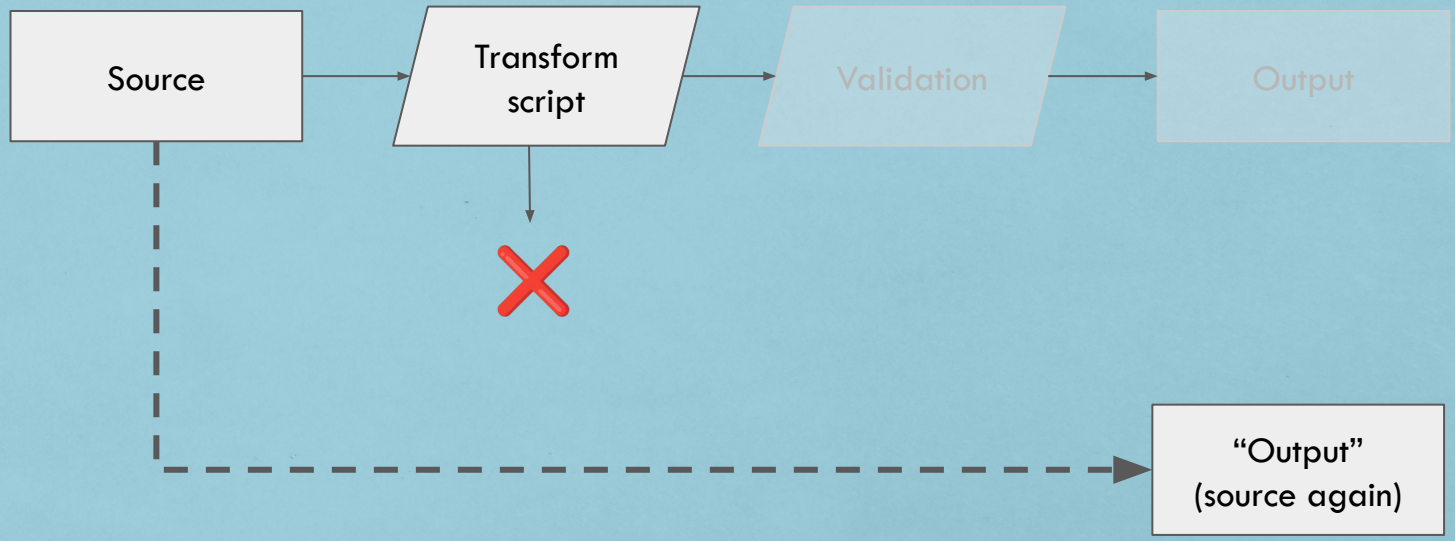
↓

Copy to build directory

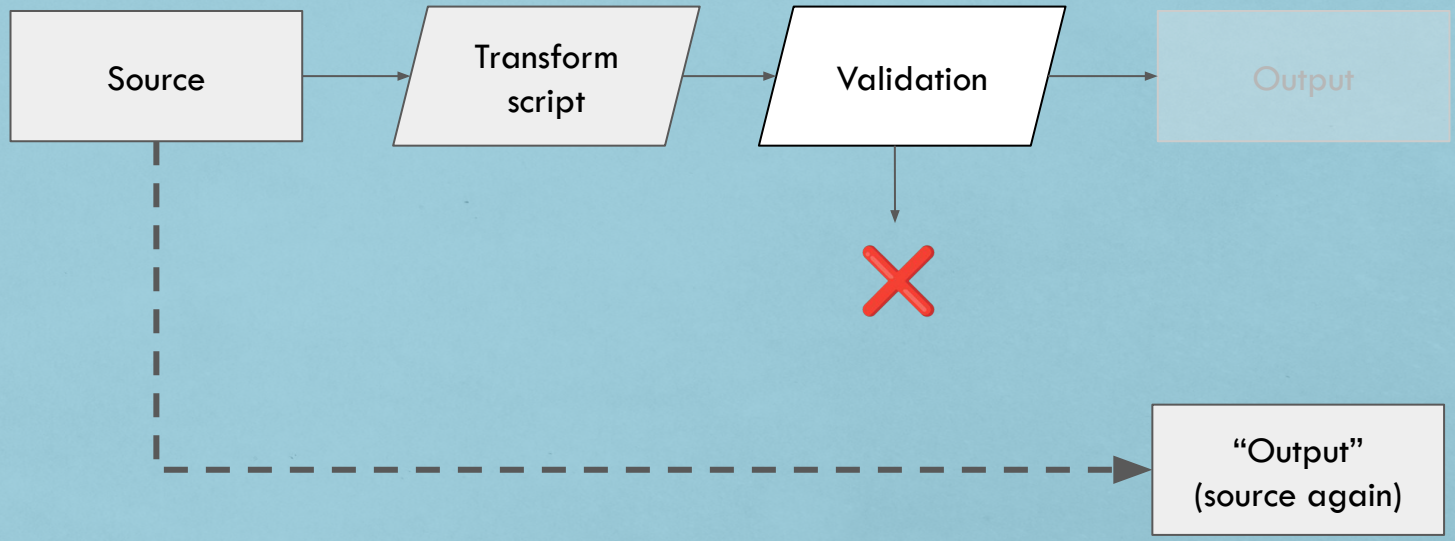
How to make each step fail forward



How to make each step fail forward



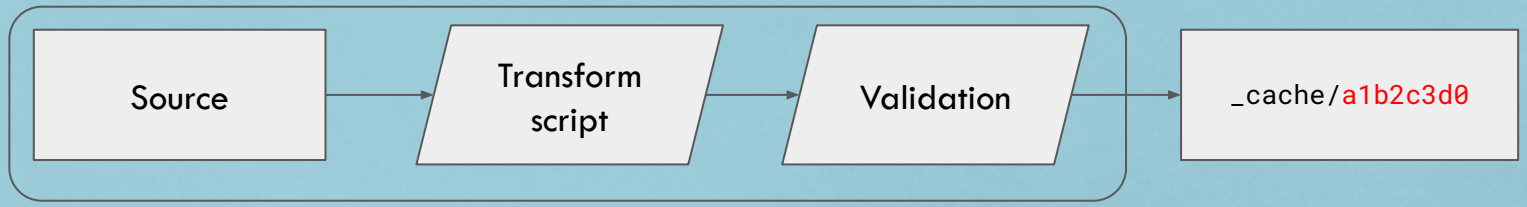
How to make each step failure-free



Fun fact: we're doing this trick with hardlinks

How to make each step *very fast*

Use a content-addressable cache



Hashes to a1b2c3d0

If source, script, and validation are the same we skip it, and reuse the cached content. Most changes result in very little reprocessing.

What technologies to use?

For any technology, assume you're somewhere in the middle of its lifespan

React.js has been around for 10 years, so you can assume it will be around for 10 more

What if you want it to last for 20 years? 30 years? Your whole lifetime?

30 year tech

The core engine is just *make* and *bash* running on a Unix. Ancient computing concepts.

Very small engine.

BUT: each transformation script can be the latest and greatest. JavaScript, Python…

If it breaks, just replace it

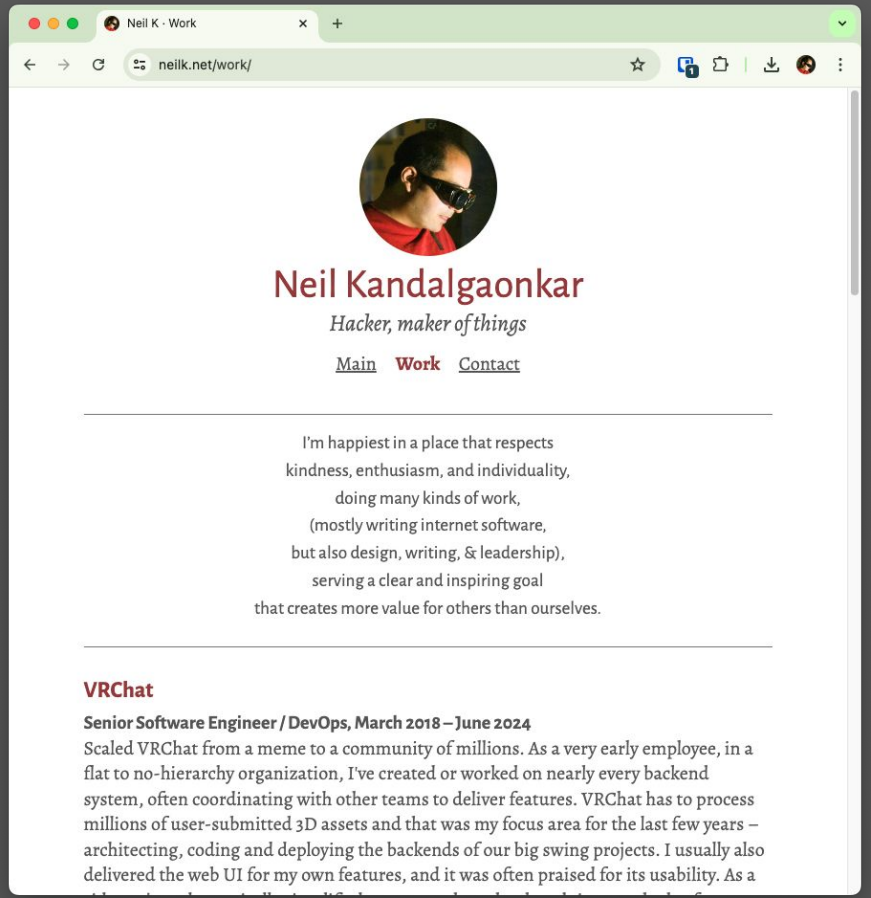
If you don't get around to replacing it, no problem

The site still works and you can still publish

Today

My personal website's CV looks like this

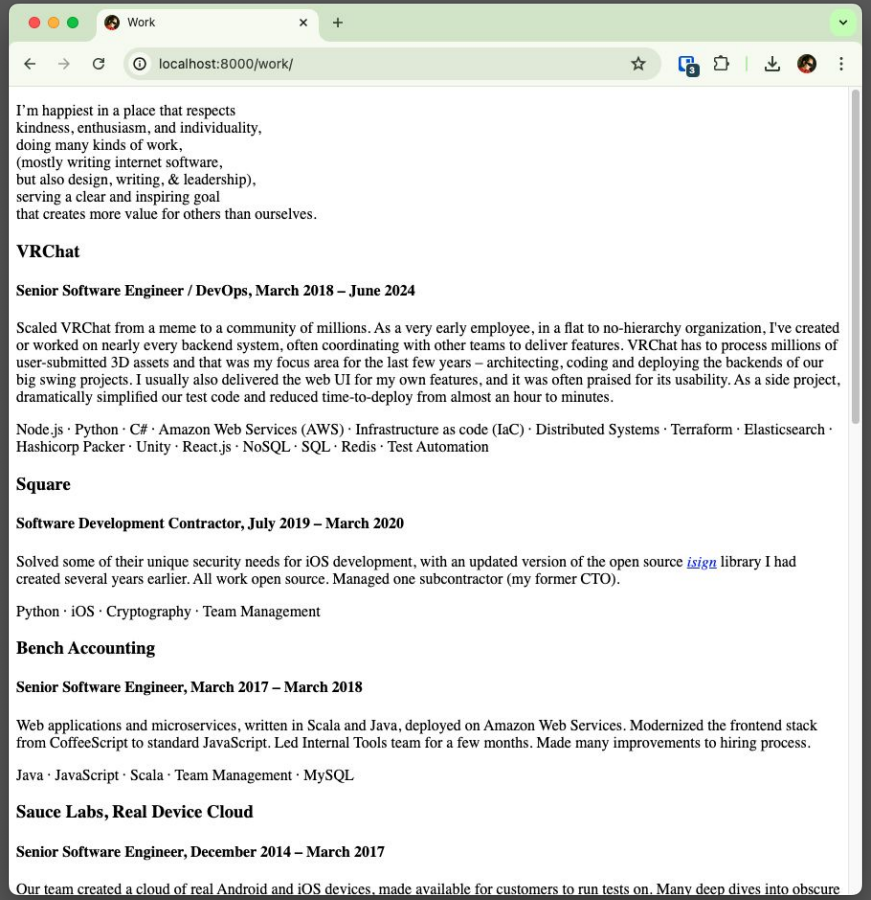
(Hint: I'm looking for opportunities…)



In a decade

Even if everything breaks, I will be able to update it, and it will look like this

(hopefully I get another job before then)



The end

Neil Kandalgaonkar
https://neilk.net/

Source code (eventually)
https://github.com/neilk/permaweb

Image credits

Wikimedia Commons

- https://commons.wikimedia.org/wiki/File:Histoire_g%C3%A9n%C3%A9rale_des_auteurs_sacr%C3%A9s_et_eccl%C3%A9siastiques_-_qui_contient_leur_vie,_le_catalogue,_la_critique,_le_jugement,_la_chronologie,_l%27analyse_and_le_d%C3%A9nombrement_des_diff%C3%A9rentes_%C3%A9ditions_de_leurs_(14597796317).jpg
- https://commons.wikimedia.org/wiki/File:Histoire_g%C3%A9n%C3%A9rale_des_auteurs_sacr%C3%A9s_et_eccl%C3%A9siastiques_-_qui_contient_leur_vie,_le_catalogue,_la_critique,_le_jugement,_la_chronologie,_l%27analyse_and_le_d%C3%A9nombrement_des_diff%C3%A9rentes_%C3%A9ditions_de_leurs_(14800512523).jpg
- https://commons.wikimedia.org/wiki/File:Old_paper7.jpg
- https://commons.wikimedia.org/wiki/File:Francisco_de_Goya,_El_sue%C3%B1o_de_la_razon_produce_monstruos_(The_Sleep_of_Reason_Produces_Monsters),_published_1799,_NGA_7502.jpg