Radical Simplicity in Technology

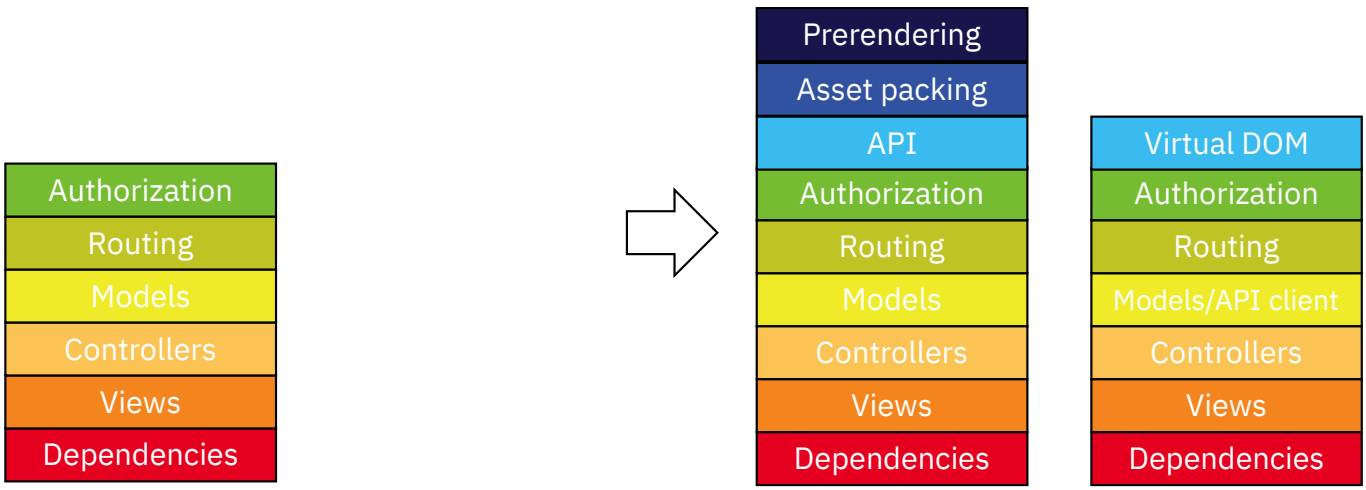
Stephan Schmidt

As developers we love complexity. We create complexity with SPAs, Vue/React, Transpiling, Typescript, Babel, Webpack, PureCSS, GraphQL, JSON, and on the backend with microservices, protobuf, Kafka, InfluxDB, or NoSQL databases. This complexity is accidental and not in the problem domain. This complexity slows us down and makes development tiresome. This complexity leads to shallow domains. Radical Simplicity makes development fast and joyful again.

This complexity needs a solution to manage it - which is often Kubernetes. And you need to run these things, and with many microservices, you need many machines. Which leads to AWS. Which brings its own complexity. Now we need frontend engineers that write React apps, backend engineers who write REST and GraphQL endpoints, and operation engineers that hand hold Kubernetes. This leads to complex test setups with mocks and local databases, Docker

images, and build pipelines that take tens of minutes to run. Rather sooner than later development is tied down by a net of complexity.

We came from something simple that worked to a massive complex thing in the last two decades.

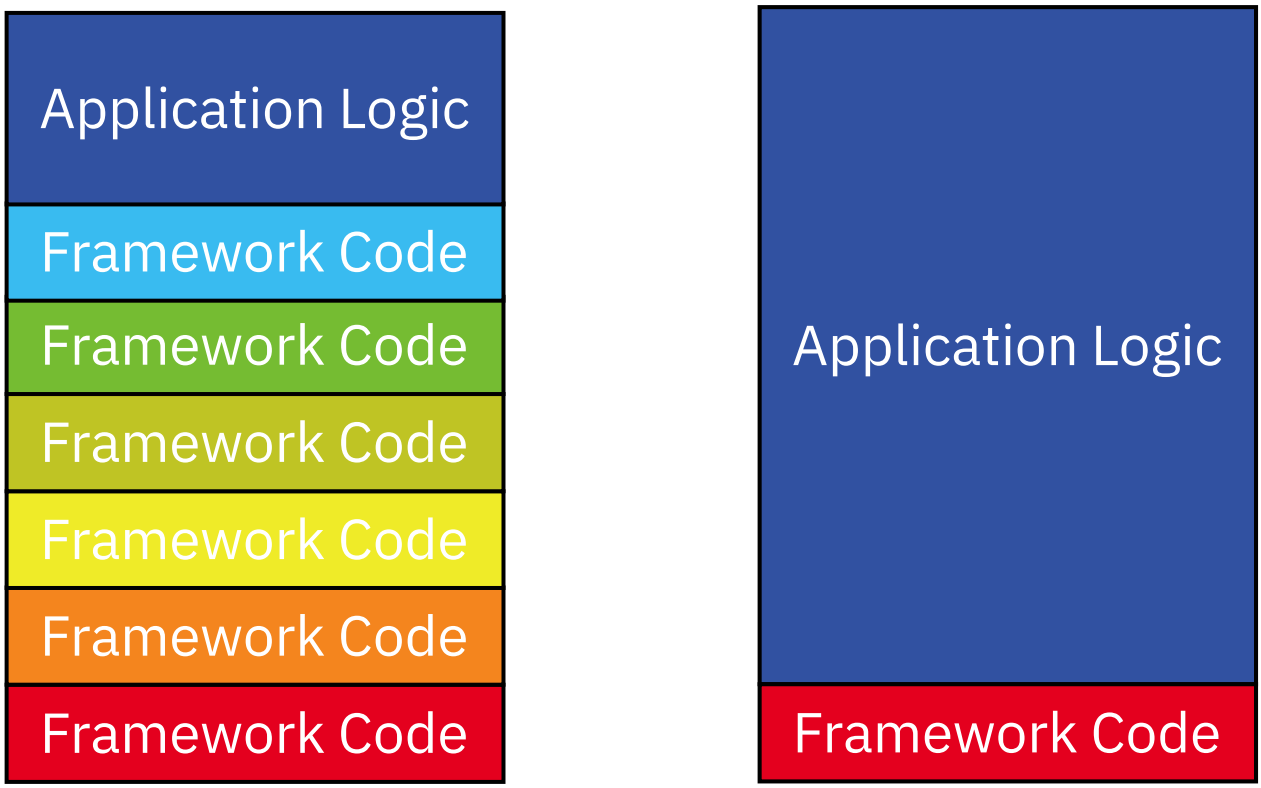


Web applications got tremendously complex over the years

It takes a lot of time for doing a simple thing, like adding a field to a form - for example a birthday field to a profile. Can't take that long? It often does. What should take minutes takes hours. And then things break, and you spend the whole day making your house of cards run again - delicately balancing every service and every version of every framework and tool on top of others.

The result of this is that the last thing developers do is writing business logic. If you look at what you are doing and count the lines, the smallest amount of time is writing real business logic, the IFs in your code. But a lot of serialization from databases with SQL into objects into JSON over

the wire into a React store to a UI. Data definitions in GraphQL, SQL, Javascript objects, and Python objects. This leads to shallow domains. These technology departments act under constant pressure from business. When you need to spend so much time on ceremonies around frameworks, tech stacks, and serialization, there is not a lot of time left to work on business logic. So domains are shallow with 90% being data transformation and IO. Apps are shallow. But what we do want, is write deep domains.



Radical simple applications have deep domains and a little code in frameworks

Why do we build complex systems and architectures? Complex architectures are a surrogate for real problems. Because we want to be

challenged and domains are shallow, developers build their own challenges with new frameworks and systems. They want to experiment with new things and more often than not want to scale technology to customer numbers that only will be there in some years - if ever. Premature scaling is one of the biggest reasons startups go bust.

The problems arising are manifold. As Rachel Kroll writes “Code runs on people. Please keep it simple.” The same goes for components and systems. Before code runs on a computer it needs to run in your head.

“If it takes an hour to figure out what’s going on, well, that’s an hour that wasn’t spent doing something else more useful and interesting."

On top of that, tech deals a lot of time with itself instead of delivering business value. Managing these many systems and components takes knowledge, and because there are edge cases there are many bugs. Combined these lead to low efficiency, to too many developers and to high costs.

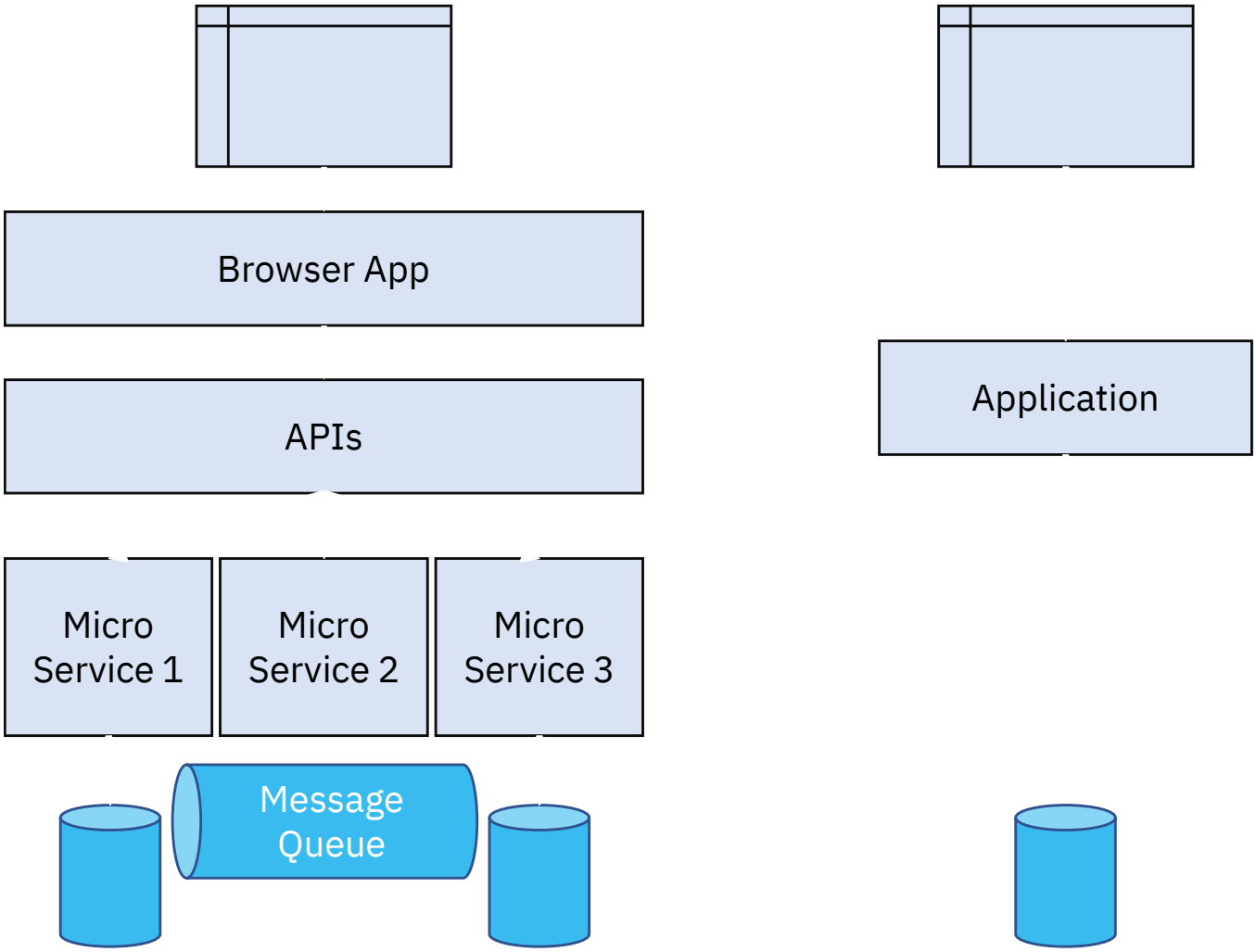
Here comes Radical Simplicity.

What is Radical Simplicity? Radical Simplicity means having as few components and moving parts as possible. Reuse technology for different purposes instead of having a new moving part for each purpose. Instead of using Postgres as a database, Druid for an event store, Redis for caching, Rabbit MQ as a message queue and Elastic for fulltext search, use a hosted Postgres as a database, for fulltext search, html caching, publish/subscribe, and an event store with TimescaleDB. This enables us to have deeper knowledge, move faster, have faster onboarding of new developers, have fewer things that can break, no upgrade planning for dozens of frameworks and components, and more developer happiness. Core to developing is getting into the flow.

Developers are much more productive when inside the flow than outside of flow. Breaking components, upgrades and edge cases of databases you need to read about break that flow.

Basecamp, the creators of Rails, do Radical Simplicity. Basecamp wrote their Hey email application with HTML and no React. Stack Overflow does this by running their service with a small amount of real hardware. It's ironic that all the AWS questions are answered by a Microsoft SQL Server running on real hardware. No GraphQL, no React, no Kafka, no Webpack, and no Kubernetes.

You can use Radical Simplicity too. Radical Simplicity is radical. A monolith that spills out HTML that is refreshed on the browser side with Hotwire Turbo with minimal Javascript. Using only managed Postgres as a database for data storage, for JSON, job handling, as a message queue, and with a columnar store as your data lake and data warehouse. If you can't resist then add Redis for caching, because Redis never breaks.



Microservice architecture versus what is really needed

Many startups with only a few customers have several microservices, Redis, Postgres, Elastic, Kubernetes, Webpack, a JavaScript SPA, REST APIs, GraphQL with Apollo and Kafka or RabbitMQ for a message queue or job server. Compare this to a radical simple setup that only uses a hosted Postgres (instead of PG, Redis and Elastic), Unpoly to render HTML on the server in a monolith and BigQuery for an analytical warehouse. A much smaller setup that achieves the same but has much fewer moving parts that need to be maintained, learned and debugged. Many fewer components need to be monitored, added to a logging server and alerts created. Do some companies need that complex setup when they have 50+ developers and millions of users? Yes. Do most of

the companies, especially in their first years, need that complex setup? No.

Standard Setup	Radical simple setup
Many Microservices	Monolith
Vue	Unpoly
Nuxt	
Webpack	
Babbel	
Javascript SPA	
REST API	
GraphQL	
Postgres	Postgres
Redis	
Elastic	
Kubernetes	

When your needs grow, add more moving parts, but slowly. Challenge yourself if you really need that new part. Radical Simplicity means extending your existing technology first. For example if you need REST/GraphQL for native mobile applications, use Hasura to automatically create REST/GraphQL from a Postgres database.

Is Radical Simplicity the same as "Choose Boring Technology"? It goes in the same direction, reducing risk and increasing efficiency. And while "Choose Boring Technology" also addresses the number of pieces ("If each tech is expensive, you should pick a few") it mainly focuses on choosing proven technology. And with proven technology you still can create a complex architecture and setup. Combining Radical Simplicity with "Choose Boring Technology" results in the best outcome. Radical

Simplicity also works with exiting technology if you have enough money to hire those few engineers proficient in it, but it's much easier to go with Postgres instead of a nifty new database and PHP instead of that funky new functional language.

There are many more positive side effects of Radical Simplicity. Features get delivered faster, founders need less developers to deliver more, the company can move faster, can easier adapt and can easier pivot to something else. Radical Simplicity gives your startup a much higher chance of success instead of failure. **If you are a founder, insist on Radical Simplicity**.

Can you love Radical Simplicity as a developer? Isn't it all about new technologies? Isn't that what brought you into programming in the first place? I think what brought you into programming is the challenge of solving problems. And the challenge should not come from learning new technologies, but from solving deep problems. With more time on your hands it's easier to solve challenging problems in the domain with new algorithms and deep features that astonish users, and not shallow features like storing data in a database from a form.

Can you do Radical Simplicity? Yes! As a founder you need to manage your CTO otherwise he or she will build a dream castle of technology. You already follow Lean Startup for maximum chances for your startup. Radical Simplicity perfectly matches Lean Startups on the technology side of things. Radical Simplicity forms a trinity together with Lean Startup for business and Scrum for process. It enables to experiment fast, deliver early and pivot if necessary. A bloated architecture ties you down and sabotages your lean startup endeavour.

As a CTO you need to build an environment where people can grow and experiment without relying on a growing tech zoo. Instead, find

challenges from business that can be solved with your inventions, your cleverness, ingenuity and algorithms and not by more toys from the shelf. My proudest coding moments were figuring out an algorithm for Towers of Hanoi as a kid and writing my own fulltext search way before there were frameworks like Lucene. Google Search has a simple UI but a deep domain. when you enter 2+2 it doesn't search for this term but prints a calculator and shows 4. This is a deep feature that users enjoy. It can be done, so do it!

Radical Simplicity takes away all the technology that doesn't deliver customer value.

Radical Simplicity leads to deep domains and deep tech.

Radical Simplicity leads to higher quality.

Radical Simplicity makes everyone happy.

Radical Simplicity makes it easy for new developers to understand a system.

Radical Simplicity makes setting up and testing easy.

Radical Simplicity puts you back in control.

Radical Simplicity wins.

About Stephan

As a CTO, Interim CTO, CTO Coach - and developer - Stephan has seen many technology departments in fast-growing startups. This is where the idea of Radical Simplicity comes from, because he has seen too many complex setups that costed money and had their problems. He taught himself coding in a department store around 1981 with

VIC20 Basic and went on to write code - and was paid for many of them - in C64 BASIC, 6502 machine code, CPC BASIC, Z80 machine code, 68000 machine code, Amiga BASIC, GFA BASIC, Blitz Basic, QBasic, Turbo Pascal, Modula-2, Oberon, Delphi, C, C++, Lisp, Prolog, Perl, Python, Java, Javascript, Scala, Erlang, Haskell, TypeScript, Go and Rust amongst others - using VIC20s, Sinclairs, C64s, Sharp Pocketcomputers, CPCs, Amigas, Beboxes, Atari STs, MSDOS machines, Windows machines, Linux - pre distro - machines, SUNs, SGIs, NextCubes, and many Quadras|iMacs|MacCubes|MacBooks|MacBookPros|iMacPros. Stephan has founded several startups and worked in small and large companies as CTO. After he sold his latest startup he took up CTO coaching. You can find him on LinkedIn