SACRA

EXPERT INTERVIEW

UPDATED
03/03/2022

Bud Parr, founder of the New Dynamic, on Jamstack's Cambrian explosion

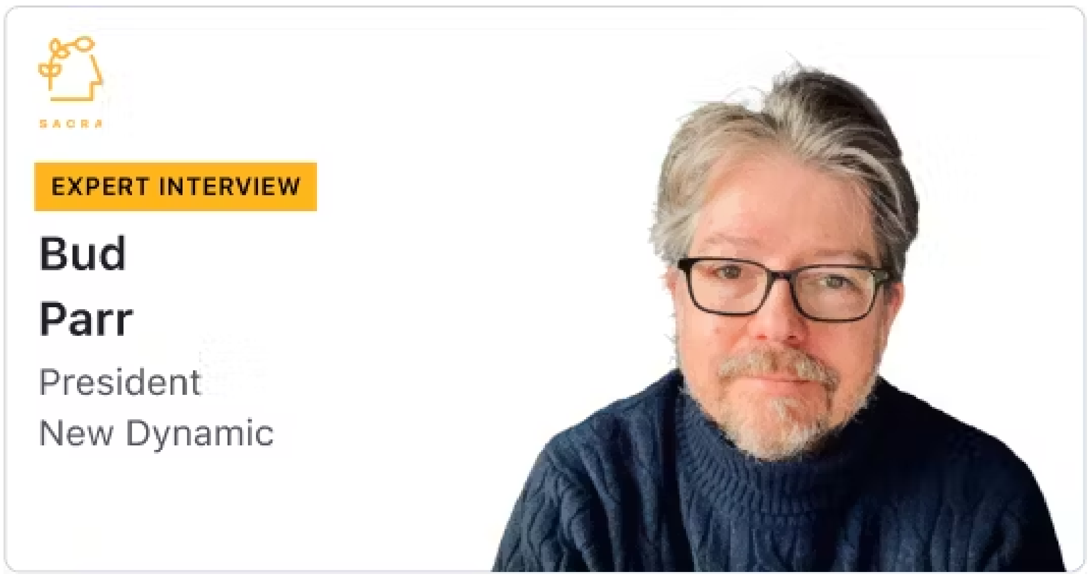
TEAM

Jan-Erik Asplund
Co-Founder
jan@sacra.com

Published on **Mar 03rd, 2022**

Bud Parr, founder of the New Dynamic, on Jamstack's Cambrian explosion

By **Jan-Erik Asplund**



Background

Bud Parr is the founder of the New Dynamic, which was working with static site generators before the term "Jamstack" had yet been coined. We talked to Bud to get a better understanding of why the Jamstack philosophy has become so popular among developers and how it's evolved since it's origins in the SSGs of the last decade.

Interview

I'd love to hear how you got started with Jamstack, and you can define your terms there as you want.

It's funny, because Jamstack has been this very helpful but loaded term. What it means has caused a lot of angst among people in the community particularly over the last year or so, because I think the sands have shifted. We went through a period of time where Jamstack was very well defined, and then

as other players have come in, technological changes have really come to the fore. The definition has shifted.
I have my own definition, and I guess you could say I'm a traditionalist as much as we could say that for something that's been around for less than a decade or so. But to my mind, it's pretty simple. Jamstack sites are those that only use the technology that they need. That means that they're decoupled. We decouple the frontend and the backend, and if we don't need a backend, we don't use a backend. That might mean that we don't have an editor. We have an editor for Git-based files and anything like that. But it's as simple as it possibly can be for any given use case. That's a very traditional way of looking at it, I think. But the idea of a Jamstack site being decoupled with frontend and backend separated is key.

The other big element for me is that the site is pre-rendered. That's part of what has shifted over the last year or so. Traditionally, a Jamstack site was one where we used the static site generator, took some content and some templates, married them and output the entire site into essentially a folder, and threw that folder online, so that when a visitor came to see the site, they were just loading that HTML. There was no logic on the server. We shifted the building of that logic from an end user to developers building the site. That's key. Over the last year or so, we've gotten more of a hybrid approach, say, Next.js and that sort of thing, where you can have some static and some server rendered and dynamic rendering of some pages and that sort of thing. It's muddied the waters. If you still think of it in terms of this decoupled architecture though, I think that's pretty safe.

But for me, we build our sites, we do everything statically generated, we push it up on our server. We're still working in a very traditional way. We've been building Jamstack sites since 2013, and I think we're pretty much the first Jamstack agency. I had some big projects that I was working on back then for a government entity and a big NGO, and they were fairly complex sites. I was able to quote them a much lower price for doing this work by using Jamstack, and at the time, these were very raw technologies, so it was a big gamble, but it worked really well. It was just so much more efficient, and these were sites that had a global audience, so being able to publish our entire website on a global CDN really was significant. We never really looked back.

At the time, my firm's name was Sonnet Media, and I really had become entranced with these technologies. I created a website called The New Dynamic. It was actually called “Static is the New Dynamic,” but we subsequently drop the “static” because that was confusing. That was to help developers find all the resources and that sort of thing, because we were moving away from a monolithic world to a decoupled world, which meant that we had to make decisions about all the different tools we were using. That site, which is still there, was really aimed at helping developers find those tools and talk about these issues, and we have a community and a meetup group and that sort of thing, all still called The New Dynamic.

What are some of the wider trends in software development that you see as having supported the initial emergence of Jamstack? Why ‘now’?

If you went to a typical website 10 years ago, you’d have to spend a lot of time waiting for it to load. Google, in particular, really began to push web performance around the time that mobile devices came into prominence. Over the same period, we started to see a more transactional web, also demanding better performance.

But I don’t think Jamstack sites would have come about if it weren’t for Github. Not only was the first prominent static site generator, Jekyll, created by a Github founder seeking a simpler way to publish a site, but Github made Git a part of every developer’s daily existence. Early static site generators primarily used Git (and practically speaking, Github or other visual tools) as a data store.

With the agency work, were there projects that would emerge that you wouldn't use Jamstack for or that wouldn't be a good fit, and conversely, where is Jamstack most useful?

I really believe in choosing the right tool for the job, and I don't have a problem if the job’s not right for my firm or if it's not right for the things that we build, then it should be built with another tool. We've had clients where they needed to be in some LAMP stack tool or monolithic tool for some reason or another -- business reasons or technological reasons. We've just said, “Goodbye.” All we do is Jamstack. There are plenty of use cases where the backend is integral to the frontend.

Now I think that that's getting even more blended. There's a lot more you can do with what are considered Jamstack technologies now. I think the use cases where you would say, “Oh, I need Drupal or WordPress” -- those might be business use cases where somebody has a preference for them, but the technological reason to use those tools I think has largely disappeared. My firm mostly does large content-heavy sites, product sites, corporate sites, some media sites and that sort of thing, and I think those are very well suited to Jamstack technologies.

What’s your take on what’s happened over the last few years as things have really seemed to take off with this more modern evolution or hybrid approach that’s emerging?

Some of it is positive, and some of it is not. I think that a lot of websites should be static websites, not shipping JavaScript to the client. They don't need a lot of infrastructure to build them. So just starting with the more negative side, I think tools like Next.js have sucked the wind out of the room. A lot of people are using them, they're comfortable using them, and there are more job opportunities to know React and that sort of thing, so it leaves some of the more traditional tools behind.

But on the other side, we still have a pretty robust set of tools that are what I would call now compilers. Instead of static site generators, they're compilers, which do essentially the same work as a static site generators. Astro, which just announced $7 million in funding yesterday, is a compiler and allows you to use React if you want, or other tools like Vue or Svelte templating and that sort of thing. They only ship the JavaScript that you need. That's fantastic. Svelte is a compiler. That tool is effectively funded, because Vercel hired the guy who created it, and it’s a wonderful tool, really brilliant. Of course, Next.js is absolutely brilliant. I think I said that I don't use it just because it doesn't fit the use cases of the work that I do, but there's a sense to me that there's nothing that you cannot do with it at this point in time, and I think that's pretty terrific.

I think the concept of Jamstack is really getting melded in. There's always going to be the people who are doing it more traditionally -- traditional in the sense of what Jamstack is. But I think it's just front-end web development, and I think that the monolithic tools are going to change and shift. We're already

seeing that. You can use Drupal headless, you can use WordPress headless. That's shifting. It's all “choose the right tool for the job.”
What are the advantages and disadvantages of headless API products like Contentful for CMS? How do you think about the trend and future of headless more broadly?

I feel like that world is just getting started. We see pricing changes all the time and not sure how their offerings will fit with the economics of our work over the long-haul. We also like that Git is very manageable and, obviously, has built-in version control and ultimately free because if our editor fails we always have local files to manage.

That said, tools like Contentful are critical for sites where one set of data may be used in multiple outputs. For example, a museum that may have its website content very tightly coupled with content it displays in kiosks or a mobile app. Contentful in particular has a rich layer of plugins that make it compelling for many. Ultimately, I think the headless CMS space will be very specialized for the most part because content modeling has gotten more complex and what works for one team or use-case may be completely wrong for another.

Javascript has had a reputation for being hard to work with -- can you talk about how writing front end code with Javascript has changed over the last 5-10 years?

For a long time JQuery dominated when browsers were not as standard as they are today. That was vital for the role it filled, but it also made a lot of us, including myself, poorer coders because we relied on it too much, particularly plugins. Then there was a push for people to learn vanilla JS, which was great, but nonetheless, that took a lot of code to get things done. That leaves room for error, so libraries work to fill that gap and give everyone a common set of patterns to work from. In some ways we’re back to JQuery in that developers get pretty far away from the core, but that’s inevitable to manage complexity and be productive.

There’s an argument that a lot of Jamstack -- however people mean that -- is a transient phenomenon in that it’s obviously a marketing term in a sense, coming out from Netlify. But it sounds like there is durability to some

aspects of the ecosystem, key aspects like static sites. How do you think about that?
I don't think it's transient at all. I don't think it's a trend. Like I said, I do think it really has become de facto front-end development for a good swath of the web, and I think that that shift is going to continue. If you go back ten years ago, performance on the web was really bad. Then when we started to think about mobile, performance became the number one factor on websites, and Google started to push that we needed performance for SEO, and that got the business people paying attention. At the time, the difference in latency between serving a static file and having to go to the server for your page logic was significant. That's probably narrowed, but there's still a difference between the two.

That really was the genesis, but now I think what's taken over is the workflow and the flexibility. I think that being in a monolithic system where you choose that system and you choose everything that goes along with it can be very confining, and particularly having to manage all of the pieces of that monolith, whether or not you need it. Those are really solid fundamentals that the web is not going to get away from, because developers -- and the people who pay them -- drive these decisions. If they're efficient, then everybody's more happy.

One idea that we've tried to wrap our heads around is that people think of Jamstack as a specific set of practices, whereas maybe it's more of a framework or approach to web development that will give birth to various products and new things over time. An analogy we're thinking of is Agile, which introduced different ways of working that led to products like Trello and others that have been durable. Does that analogy make sense to you?

Well I think Agile and Jamstack methodologies go hand-in-hand with one another. Jamstack workflows are literally very agile because it's dead simple to create a branch of your site, make changes, consort with stakeholders and, when ready, merge those changes into your production site.

My firm works with organizations over a long period of time to maintain their websites. The time we spend on their sites is spent helping them improve their site, not doing tedious work just to keep the site running, which allows us to price such that

cost, profitability, and client outcomes are not so much at odds with one another. Prior to using Jamstack I would have clients paying maintenance fees and not see much else because just keeping the site going ate up their budget. We still manage some old LAMP stack sites and I cringe whenever we get notifications about the site being down because of an influx of traffic.
An entire ecosystem has emerged to support decoupled architectures where every part of the site is a thin layer that can be relatively easily replaced, including the editing interface, e-commerce tools. We've seen hundreds of "Headless CMS" tools released, creating a wide variety of choice for any use-case.

So, one doesn't have to choose Wordpress, say, and Woo-commerce and a grab bag of Wordpress plugins that may or may not play well together. We can even use just the Wordpress editing interface if we want to.

Jamstack and Vercel/Netlify are widely known for the ability to get up and running with projects quickly. Can you build fully-featured apps as easily? What prevents you from building your whole stack on e.g. Vercel, and where are the breakpoints? At what point might you move from Vercel/Netlify/Jamstack to building direct on AWS, or on Heroku?

I think that those services are much more than just a quick solution. We've been big users of Netlify from the very beginning and find their workflow mission critical. We use deploy previews every day with clients. The concept of atomic deploys–where only changed pages are pushed to the server–are critical in keeping deployment-times fast, and then the easy integration of serverless functions really helps us keep our repos straightforward and easy to manage. I'm sure Vercel has much of the same features, but we've never felt a need to look elsewhere.

I think these firms are pushing innovation. Vercel, of course, is behind the hugely popular NextJS project. I find AWS a total pain to use. It's useful, obviously, but not for our work.

I'd love to ask about your stack. I'm curious if you use Vercel or Netlify and also as far as CMSs and other APIs. Do you have preferences where you'll always use

Contentful or X for your CMS, or only use Algolia for search?
Yeah, but we do try to keep our options open. The original idea of static site generator was that we would manage our content in Git. We still do that, for the most part, but not 100%. But the idea of managing your content in Git is that it's free basically, because all your content lives within your repo with your code base and that sort of thing. Some will argue that that's a very bad idea. But from my standpoint as a business who manages many, many websites, that's a wonderful attribute. Having everything in one little package is wonderful for me from a business standpoint. I also get free version control included with that. It's just very compact and easy to manage.

If my editor fails, I still have my local copy to manage. Or it makes it very easy for a developer to step in and fix something or to update copy for a client. Instead of going into the CMS and figuring out where they are, because we manage so many sites, we can just pull down the repo, edit the content and push it, and that's very simple.

The editors in that space, there are not that many of them. We use Netlify CMS, we use Forestry, we use CloudCannon. They do a good job, but it's a very difficult thing to do to manage on the frontend.

The API-based CMSs are numerous. We don't tend to use them so much right now, but it's clear to see tools like Sanity and Contentful are really just doing fantastic things. I think for our own purposes, Contentful is probably not at the right price point. And I think that that's one thing about the API-based CMSs that we're seeing, is that they still don't quite know how to price their product. It's a very difficult thing to price, right? Particularly if you're an agency and you've got a client who is going to absorb these costs, and they're used to WordPress, which is free for all practical purposes, or Drupal or something like that, and then you're asking them to pay hundreds or thousands of dollars per month just for that editing interface that used to be free. I wouldn't want to be the guy responsible for trying to figure out how to monetize that.

I think they're doing a good job of it, but we see the prices shift over time, and it's a little bit hard to know where they'll be in five or ten years' time, and I try to make my decisions based on what I see in the long term. That uncertainty gives me pause about the tools. But clearly, the messages for some of the better tools are that they're wanting enterprise customers. For people who manage many, many sites, the economics of that don't work out so well, because although we have enterprise clients, we're not in that realm where thousands of dollars a year for CMSs makes sense to us.

That goes to something we're curious about, which is the comparison to Automattic and whether a bunch of headless CMSs are going to take over, when something like 40% of the web is on WordPress. We use Prismic, and I can figure out adding content, but I'm not sure if everyone in the world can the same way they can on WordPress. How do you think about that long-term, especially since you work with media companies and other places where you have non-technical folks having to add content?

I think that the editing interface for a website is always a huge abstraction for content people. They think in terms of "I've got a page, and on that page I have some body of content or whatever." To those of us who do content modeling, we break out all those pieces where they can be reused in different places and that sort of thing. That abstraction creates huge problems for people, just cognitively. Then as your site grows in complexity and you're using your content in many different ways, that's a really difficult thing to manage.

I do think that there's a lot of effort on the part of, say, Stackbit, TinaCMS, CloudCannon is getting this feature, and I think Prismic has Slices. We see this trend -- and I think Slices is this -- where content editors are allowed to go onto the page and visually edit that page, which puts that more into the realm of, say, a Squarespace or something like that. I think that that's not going to be a panacea. I've seen people editing Squarespace, and I think they're a nightmare, and the results are really bad. Where you really get the best results is where you have humans and they're helping with this integration between the two and where you can just make everything really tight and really reuse your content over there. But that's not everybody.

I think that people are making a good attempt at doing that. I think WordPress has a degree of familiarity for people, and there's that big box of content that people can use. But we're doing a site right now -- a migration from WordPress -- and the sorts of things that these people had to do in WordPress to get it to look how they wanted in the frontend are just terrible. It's just not stable. It doesn't make any sense. That's a disconnect, but I'm very happy to see people trying to deal with it.

I'd love to hear more about what you see on the backend pre-moving to Jamstack and what issues you see folks having, whether it's with WordPress or other services. I remember working in WordPress instances that had way too many files, and any clicking you did, you'd have to wait five seconds for the cursor to recognize that you had clicked.

Yeah, it's always been a real disconnect. I used to use a tool called ExpressionEngine, which is still around, and a tool that arose out of ExpressionEngine is Craft CMS. Those are both LAMP stack tools, but they've done a really wonderful job. I think Craft is actually doing amazing work. I don't use it on a day-to-day basis now, but it gives people just the kind of flexibility so that a developer and/or content modeling person can go in and design that CMS for their use cases. Then it still comes down to some training and good labeling and/or user manual, and it depends on the use case. That's evolved outside of Jamstack, I think, as well.

Right now it's a lot easier to go and build a headless API-based CMS than it is to go and build out a tool like Craft CMS, which is monolithic, it's everything -- although you can use Craft with an API, I think. But if I'm a company and I say, “I'm going to go build a tool for the web,” I'm going to choose building a headless CMS. I'm not going to choose building a monolithic tool. Because it's a more fluid thing to build, you can really spend a lot of effort into making that interface work well for you. I think that Sanity has something called a river or a stream of content or something -- I forget the name of it -- where you can work with real-time content and that sort of thing. It's amazing what you could do when you're just focused on building that interface for people and a way to serve that for them. I think that evolved. Again, I think that with these on-page editing systems that are coming about, it's finally starting to get to a good place, but there's always going to be that disconnect.

You mentioned that you think about the next five or ten years in terms of making decisions. What do you think Jamstack will look like in five years, and are there any big

important trends you see that are moving things in that direction?
Other than the fact that I really felt strongly about Jamstack from as soon as I discovered it and I really never looked back, beyond that, in terms of trends within that trend, I've not been very good at predicting things. I didn't really know where, say, serverless functions were going to go and that sort of thing. I didn't predict that. In short, I didn't predict how good things were going to be. I really thought that the best thing that we could hope for was a good editing interface.

When I started, we had a tool called Prose.io. A former Drupal firm built the new version of HealthCare.gov after the old version failed spectacularly. They built that in Jekyll, and they built an editing interface for it. It was really meant for that site, for that instant in that time. It was really hard to use, but it was a little bit better than, say, sending your clients to GitHub, which I actually tried. But that's the best I had hoped for at the time -- that we would have a good editor.

I had no idea that we were going to have things like serverless functions and be able to pull all of this functionality into our sites. I had no idea how robust the API economy was going to be. I had no idea how important deployment was going to be. We use Netlify -- we've been using them from day one. I'm aware of Vercel, and I think that the feature set is pretty much the same, but I don't know Vercel as intimately. But we use the deploy previews that Netlify provides for us every day with clients. The concept of atomic deployment, where they're only pushing to the server pages that have changed on our website, saves us a lot of headaches. The ability to package up our serverless functions right with our repo and push them out to Netlify is fantastic. I never really dreamed that that sort of thing would be around now.

So I would hesitate to say where I think things are going to go in the next five years. I think that we're going to continue apace with an improved editing experience. Oddly, it's interesting to me that we're seeing the build tools getting funded in the open source space with Astro's $7 million funding for what is essentially a static site generator. There might be more coming in the future.

You had mentioned Automattic and Jamstack, and I think the closest company to that is really Gatsby. They built a tool that

became enormously popular, which has some WordPress-type of features where there's a rich set of plugins, and it makes it easier for people to come in and add functionality without at least a huge depth of engineering, though it's still more complicated than WordPress. Then they're hosting those sites as well, which is what Automattic does. I have no idea how that's going to work out. I don't understand the economics necessarily of a Gatsby or Netlify or Vercel, because there is always to my mind these big companies looming in the background that knew a lot of that work and can probably pick up some of that functionality and maybe obviate these firms. I don't know.
But I do know that what's really wonderful is all of the innovation that we've seen, and then people are finally starting to fund open source and see the value of open source, and we're seeing this combination of open source and private tools, or where a private company is providing part of the service but open sourcing the other part of the service, which makes sustainability for open source. I think that that sort of thing is going to continue in the future. I'm not sure what else though. There are probably some innovations, though, that Matt Biilmann is creating in a back room somewhere that I have no idea about, and I'm going to be very pleasantly surprised when they tell us about it.

If you think of Gatsby as like WordPress or Automattic, could you think of Vercel and Netlify as Heroku? Then also, what kind of distinction do you see between Astro and Gatsby or maybe Astro and Jekyll? Do these all make sense to lump into a category in some respect?

I don't think to say that Gatsby is like WordPress is exactly right, but I do think Gatsby/Automattic is a very apt comparison.

In terms of tools like Astro and where they fit in, say, with Jekyll, of course Jekyll is not really in use that much. It's probably in use in a legacy way, but now we have tools like 11ty and that sort of thing. I'm not really sure how those things will shake out or what the differences should be. I think Gatsby is really trying to be -- they would probably kill me for saying this -- like the entry-level React tool that's going to allow people to build things way beyond what they could otherwise build. I think Astro will always be a relatively expert-level tool for somebody like myself, where they want to have control over

every part of the development process and that sort of thing -- although they might develop a plug-in architecture, too. I don't know.
We use a tool called Hugo, which is a pure static site generator. We use it because it's very powerful and very robust. We ship the binary with our sites and that sort of thing. We could just build that same site a decade from now, and that kind of stability or anti-fragility is very important to us and enterprise clients or mission-critical clients. There's a lot to choose from in the tool set.

How well have traditional cloud providers like Amazon and Azure adapted to Jamstack? How has the way developers utilize these cloud services changed in Jamstack?

I think they've not really managed to create a developer experience that Netlify and Vercel have. The smaller players also have the advantage of using multiple providers for their infrastructure, so instead of, for instance, just using Google Cloud to serve your site, your site may shift between them, which creates a layer of stability and may end up performing better.

Disclaimers