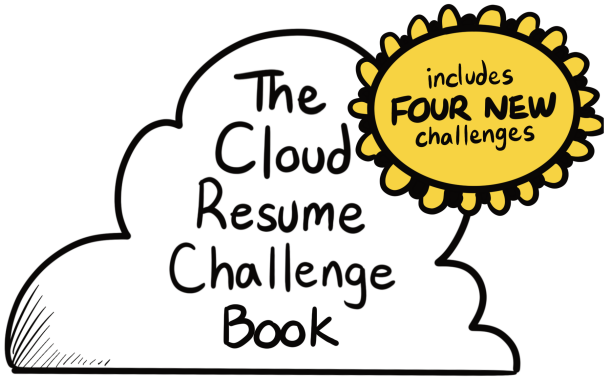


The Cloud Resume Challenge Book

a proven way to level up your tech career

by Forrest Brazeal

Some elements of "Getting certified" (Part 1) and "Networking your way to a job" (Part 3) are adapted from material originally published on the A Cloud Guru blog. Used with permission.

Prologue: The Greatest Resume I've Ever Seen

When COVID-19 hit, Daniel Singletary was already pretty much fed up with his job. As a commercial and residential plumber in metro Atlanta, he pulled 11-hour days working some of the dirtiest, stinkiest problems in the country.

Take, for example, the day he got a call about an unexplained odor in a suburban shopping strip. Daniel and a coworker headed to the scene. There was no mistaking the smell. It was sewage, and it was raw.

On a reconnaissance trip to the restrooms, Daniel noticed something odd: a current of air was flowing around the base of the toilets. When he levered a commode off the floor, he staggered backwards, hit by a blast of noxious wind. "Imagine," he wrote later, "a leaf blower blowing sewer gas in your face." Not only is this unusual, it shouldn't even be *possible.* Sewer pipes don't blow air. And yet somehow, this entire shopping strip was passing gas.

There was nothing to do but work the problem step by step. Over the next three days, Daniel and his partner worked over the building from opposite ends, unsealing and re-fitting every plumbing fixture they could find. Eventually, they narrowed the source of the mystery airflow down to two possible locations: a hair salon and a restaurant.

This is where the problem got extra tricky. How do you troubleshoot the plumbing in a restaurant without, you know, closing down the restaurant? Eventually Daniel had a bright idea: a smoke test. Literally. Up to the roof vents he went, armed with smoke bombs. His reasoning: "Wherever sewer odor can get in, so can smoke … except we can see smoke with a flashlight."

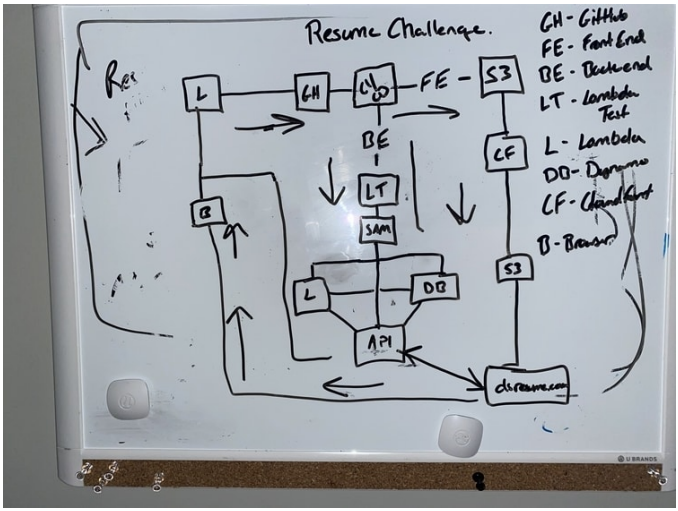
Sure enough, following the clouds of smoke cleared up the mystery. Someone had tied the vent hood for the restaurant's stove into the sewer system, forcing air into the pipes. The immediate problem might be solved, but Daniel's desire to leave plumbing, maybe even to a job where he could choose what to smell, was only growing.

It was around this time that Daniel and I met. His roommate, an IT worker, had shown him a blog post I wrote about something called the Cloud Resume Challenge. The challenge was designed to help people get their first job in cloud. It came with a promise: host your resume in the cloud, and I'll do

whatever I can to help you get it in front of the right people.

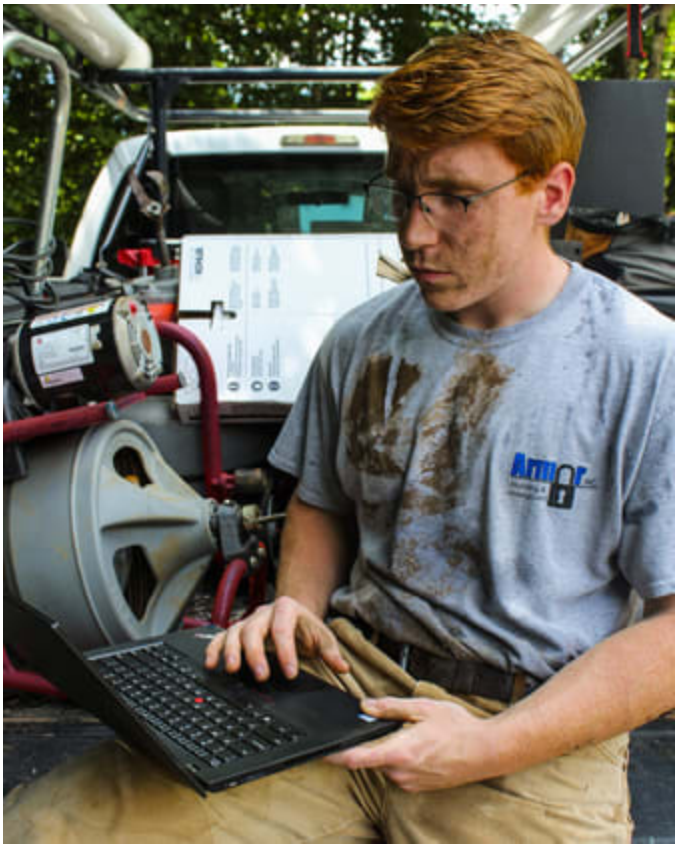
Of course, there were some caveats. Your resume had to have an intro-level AWS certification on it. And the project spec required you to get your hands dirty with source control, CI/CD, Python, front-end Javascript, back-end cloud services, and more - the full cloud stack. Basically, if you could pull off this project, you'd be doing things that even some professional cloud engineers have never done.

Daniel wasn't a professional cloud engineer. He'd never even seen YAML before. So he did the first thing that occurred to him: he went out and bought a whiteboard. He called what he drew there to help him make sense of the project an "engineered print," just like he'd used countless times as a plumber. He didn't know he was drawing a cloud architecture diagram.



Daniel's whiteboard (courtesy of Daniel Singletary)

Over the weeks that followed, Daniel forced himself to sit down at his computer after those 11-hour days. He grappled with Python and Javascript in the back of his work truck. One minute he was wrangling sewer pipelines - the next minute, CI/CD pipelines.



Let's not ask what's on his shirt. (Courtesy of Daniel Singletary)

Finally, miraculously, he completed the challenge. I reviewed his code myself; it was solid. And you can check out his resume page for yourself. It's not flashy, but it contains probably the greatest combination of credentials I've ever seen. Daniel is

licensed in backflow prevention, journeyman plumbing, residential / commercial septic installation … and oh yeah, he's earned four certifications on AWS.

One of the most important requirements of the challenge is to write a blog post about your learnings. Daniel's post, A Plumber's Guide to Cloud, went viral on LinkedIn and was shared more than 200,000 times. That got the attention of hiring managers.

And barely a month later, he started his first tech job as a cloud DevOps engineer. From all reports, he's killing it.

Now, to be clear, Daniel's success didn't come from some magic contained in the Cloud Resume Challenge. It came from Daniel. It came from his hard work, his perseverance, and (just as importantly) from the skills he'd mastered in his plumbing days.

For instance, if I'm a hiring manager on an infrastructure ops team, I'm taking away quite a few things from Daniel's story of the leaf-blower sewer:

He knows how to troubleshoot.
Given a fairly enormous problem (this entire building stinks), Daniel didn't flail around or try random stopgap measures. He narrowed the

problem down to a root cause (the blowing air) and then methodically eliminated possibilities until he found the solution.

He knows how to collaborate.
Daniel worked closely with a colleague throughout the onsite engagement, dividing and conquering to speed up the "debugging" process. Pair programming will feel natural to Daniel because he's used to having a sounding board to work out complex problems.

He knows how to test and observe.
Until I met Daniel, I actually didn't know that "smoke testing" was a term with a literal smoke-related meaning. I'd always heard it used in the context of software tests. Daniel used a tracing technique to follow a problem to its source, instead of just making random guesses.

He understands business continuity.
Daniel had to keep several businesses online and operational while conducting his diagnostics and resolving the problem. He couldn't simply flip a switch and cut the building's water supply for a few days while he figured out what was happening.

Put simply, Daniel had rock-solid real-world ops skills, better than that of most university computer science graduates I've met. What the Cloud Resume Challenge did was fast-track Daniel to the

hands-on technical abilities he needed to build on that wealth of trade experience and vault into the cloud.

Daniel's story is unique, but he's not alone. In the past 16 months, thousands of people have attempted the Cloud Resume Challenge. The majority of them don't get far; it's not easy, that's what makes it worth doing.

But those who persevere have seen incredible results. Career-changers have made the jump to cloud jobs from fields as diverse as food service, HR, retail, and recruiting. And even more people have used the challenge to polish up their existing IT skills and leverage better jobs within the industry.

You'll hear many of their stories in this book. Like them, you bring skills and perspectives that the tech industry needs. And if you commit to follow the path I'm about to show you, there's no limit to what you can achieve.

Ready? Let's do this.

Part 1: Cracking The Cloud Resume Challenge

In April 2020, as the first wave of COVID-19 descended, I found myself like millions of other people around the world: stuck at home, feeling bored and helpless.

I did have one huge privilege, though. As a tech worker, I could still do my job from home. Friends and family were telling me: "I've been thinking about changing careers and going into IT. Maybe doing something in the cloud, like you. Where should I start?"

I started jotting down some ideas: a blueprint, a roadmap of key cloud skills. Things that would make me say "Of course I'd hire that person!", never mind their previous work experience. What emerged a few days later would change a whole bunch of lives, starting with my own.

The challenge itself

Here are the sixteen steps of the original Cloud Resume Challenge, more or less as they first appeared in my "Cloud Irregular" newsletter on April 27th, 2020.

1. Certification
Your resume needs to have the AWS Cloud Practitioner certification on it. This is an introductory

certification that orients you on the industry-leading AWS cloud – if you have a more advanced AWS cert, or a cert on a different cloud, that's fine but not expected. No cheating: include the validation code on the resume. You can sit the CCP exam online for $100 USD. If that cost is a dealbreaker for you, let me know and I'll see if I can help. A Cloud Guru offers exam prep resources.

2. HTML
Your resume needs to be written in HTML. Not a Word doc, not a PDF. Here is an example of what I mean.

3. CSS
Your resume needs to be styled with CSS. No worries if you're not a designer – neither am I. It doesn't have to be fancy. But we need to see something other than raw HTML when we open the webpage.

4. Static S3 Website
Your HTML resume should be deployed online as an Amazon S3 static website. Services like Netlify and GitHub Pages are great and I would normally recommend them for personal static site deployments, but they make things a little too abstract for our purposes here. Use S3.

5. HTTPS

The S3 website URL should use HTTPS for security. You will need to use Amazon CloudFront to help with this.

6. DNS

Point a custom DNS domain name to the CloudFront distribution, so your resume can be accessed at something like `my-c00l-resume-website.com`. You can use Amazon Route 53 or any other DNS provider for this. A domain name usually costs about ten bucks to register.

7. Javascript

Your resume webpage should include a visitor counter that displays how many people have accessed the site. You will need to write a bit of Javascript to make this happen. Here is a helpful tutorial to get you started in the right direction.

8. Database

The visitor counter will need to retrieve and update its count in a database somewhere. I suggest you use Amazon's DynamoDB for this. (Use on-demand pricing for the database and you'll pay essentially nothing, unless you store or retrieve much more data than this project requires.) Here is a great free course on DynamoDB.

9. API

Do not communicate directly with DynamoDB from your Javascript code. Instead, you will need to create an API that accepts requests from your web app and communicates with the database. I suggest using AWS's API Gateway and Lambda services for this. They will be free or close to free for what we are doing.

10. Python
You will need to write a bit of code in the Lambda function; you could use more Javascript, but it would be better for our purposes to explore Python – a common language used in back-end programs and scripts – and its boto3 library for AWS. Here is a good, free Python tutorial.

11. Tests
You should also include some tests for your Python code. Here are some resources on writing good Python tests.

12. Infrastructure as Code
You should not be configuring your API resources – the DynamoDB table, the API Gateway, the Lambda function – manually, by clicking around in the AWS console. Instead, define them in an AWS Serverless Application Model (SAM) template and deploy them using the AWS SAM CLI. This is called "infrastructure as code" or IaC. It saves you time in the long run.

13. Source Control
You do not want to be updating either your back-end API or your front-end website by making calls from your laptop, though. You want them to update automatically whenever you make a change to the code. (This is called continuous integration and deployment, or CI/CD.) Create a GitHub repository for your backend code.

14. CI/CD (Back end)
Set up GitHub Actions such that when you push an update to your Serverless Application Model template or Python code, your Python tests get run. If the tests pass, the SAM application should get packaged and deployed to AWS.

15. CI/CD (Front end)
Create a second GitHub repository for your website code. Create GitHub Actions such that when you push new website code, the S3 bucket automatically gets updated. (You may need to invalidate your CloudFront cache in the code as well.) Important note: DO NOT commit AWS credentials to source control! Bad hats will find them and use them against you!

16. Blog post
Finally, in the text of your resume, you should link a short blog post describing some things you learned while working on this project. Dev.to is a great place to publish if you don't have your own blog.

Why the challenge works

Unlike most technical resources you'll find, the Cloud Resume Challenge isn't a tutorial or a how-to guide. It's a project spec, and not a very helpful one at that. It tells you what the outcome of the project should be, and provides enough structure that you don't go too far off the rails, but other than that - you're on your own!

At first glance, this seems like kind of a mean thing to hand someone with no previous IT experience. Wouldn't beginners need their hands held every step of the way? And so my initial vision, that the challenge would help people get their first job in tech, was greeted with skepticism:

Not picking on Cesar here, but it would have been smart to take the over on his prediction.

A year later, thousands of people have attempted the challenge. Hundreds have completed it. Dozens have been hired into their first cloud jobs, from fields as diverse as banking, plumbing, recruiting, and HR. Never bet against raw human determination when dollars are on the line.

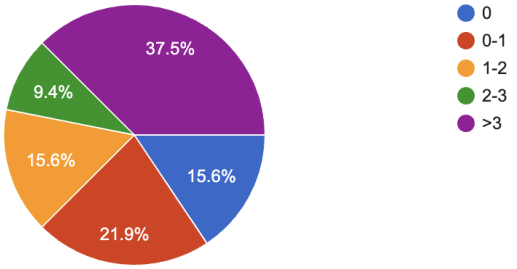
I didn't choose the challenge format arbitrarily, though. I designed it after some of the best learning experiences I've ever had: the capstone projects in my college and graduate-level computer science classes.

No, I'm not saying you need a master's degree in computer science to get a job in cloud. But good university classes gave me a gift, something most vocational training courses haven't adopted: open-ended projects that told me roughly the direction to go, then forced me to figure out how to achieve the objective as I went along. Because I wasn't just carrying out rote instructions, the learning wasn't pre-digested. The learning *came from me.*

Looking back, most of the good learning I had in college came from those types of projects. And whether or not you have a degree background, I want to give you that experience as well.

The biggest surprise to me has actually been not how many career-changers have tried the challenge, but how many *working engineers* have given it a shot. Here is a breakdown of prior experience as reported by the 1500-plus community members in the Cloud Resume Challenge Discord server:

Number of years of prior professional IT / software development experience by challenge participants



That's right - almost 40% of challengers are coming into this with more than 3 years of professional IT experience. It turns out there are good reasons (wholly unanticipated by me) why the Cloud Resume Challenge has value for people well-established in their tech careers, and we'll get into that later.

But for now, just think of it this way: if the end result of the Cloud Resume Challenge is impressive enough to make experienced IT people say "I want

that on my resume", it is *really* going to make you stand out as an entry-level job-hunter.

If you give this project a try and realize that you hate it, or you're just not interested, you'll have learned a valuable lesson about whether or not you really want a career in the cloud – because these are the types of problems that real cloud engineers and developers really work on. It's not a contrived toy exercise.

But I believe that if you can, in good faith, complete the Cloud Resume Challenge, you will already have more useful skills than a lot of people who graduate from university computer science programs. Specifically, you will understand something about:

- Full-stack software development (the static website and Python pieces)
- Version control (the Github piece)
- Infrastructure as code (the CloudFormation / SAM piece)
- Continuous integration and delivery (connecting GitHub Actions, SAM, and AWS)
- Cloud services and "serverless" (Lambda, API Gateway, DynamoDB, Route53, CloudFront, S3)
- Application security (IAM, S3 policies, CORS, API authentication/authorization)

- Networking, as in the way computers talk to each other (DNS, CDNs, the whole "cloud" thing)
- Networking, as in the way people talk to each other (the blog post, the Discord community - this is probably the highest-value step in the whole challenge because of the professional doors it can unlock for you, and we'll talk about that in more detail later on.)

I'd recommend preparing a story about your experience with each of these bulleted items to use in job interviews. The reality is that "baptism by fire" is the best way to learn, because there's nothing like pounding your head against a DNS problem for two days to burn it into your head in a way that cramming for a test never could.

Moreover, you'll have learned by doing, because I didn't give you enough instructions to figure any of this out without going down some late-night rabbit holes.

Most importantly, you will have demonstrated the number-one skill in a cloud engineer's toolbox: the ability to **learn fast and google well**.

I've spoken with dozens of hiring managers over the history of the challenge, and all of them insist that "ability to pick up new things quickly, in a self-motivated way" is their most highly prized quality in a team member, more than competence on any specific technology. To be perfectly blunt, the tools and services you've used in the challenge may all be obsolete in 5 years. But your ability to glue them together into something that works will remain. It will get you interviews and build your confidence that you can thrive in whatever technical paradigm comes next.

Nana's Story: From Tech Support to Cloud Native Engineer

Nana, based in New York City, already had some technical experience as a help desk administrator, but needed something more relevant on his resume to achieve his dream of getting a full-time cloud engineering job.

Nana says it took him about **4 weeks, coding about 5 hours each day** to complete the challenge. "It was the kind of project that established full-stack as well as DevOps knowledge."

About his interviewing experience, Nana noted: "The resume challenge helped me demonstrate important skills, particularly understanding of CICD automation, DNS (and how the internet works), SCM/git, and serverless architecture. Hiring managers seemed to appreciate rounded developers who also had some experience with agile and the DevOps culture."

Nana used his first round of interviews, mostly with Fortune 500 companies, to identify remaining gaps in his knowledge, then took a 4-month break from interviewing to pursue more hands-on learning (including the ETL challenge provided later in this book!). About six months after initially starting the Cloud Resume Challenge, he landed a job as a cloud native engineer at Accenture.

For future challengers, Nana offers this wisdom: "Get hands-on, and don't rush it!"

Breaking down the challenge

Here's the biggest mistake I made when writing the original Cloud Resume Challenge: it was set down in stream-of-consciousness order. While the steps are roughly sequential, I didn't think rigorously enough about exactly what order they should be in. For that reason, many, many people have asked if they can complete the steps in some different order that seems more sensible to them.

The short answer to that question is always YES. There is no wrong way to attack the challenge, as long as you are doing your own work. That said, the challenge does break into four fairly discrete chunks of sequential steps, which can be stacked in almost any configuration:

- Getting certified (step 1)
- Building the front end website (steps 2-6)
- Building the back end API (steps 7-10)
- Automating everything and setting up continuous integration (steps 11-15)

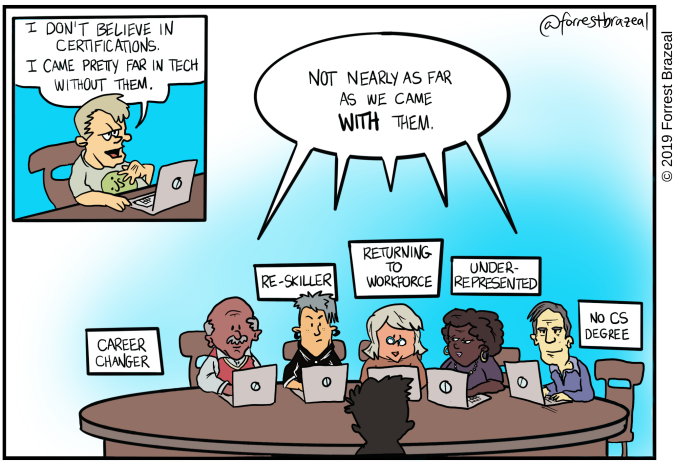
Let's talk briefly about the strategy and reasoning behind each of these chunks, as well as the blogging and code review requirements, and then I'll lay out my personal recommended roadmap for the challenge.

Getting certified

Certification is the only step that you can complete literally at any point in the challenge. You can do it up front, save it til the end, or study for your cert while you are completing other steps. In that sense, it's disconnected from the rest of the challenge.

But let’s get real about certs for a minute. I’ve yet to meet a hiring manager who believes that IT certifications guarantee technical ability. (And let’s be honest, you wouldn’t want to work for someone who believed that.) A cert is just a piece of paper, after all — and that’s only if you print out the PDF. It doesn’t magically give you the experience you need to succeed in the real world.

And yet, cloud certification has been helpful in my engineering career, and I encourage it for everyone looking to get a foothold in the industry. Certs are powerful tools, as long as you understand what they are and are not good for.



Certs give you a comprehensive knowledge base

You may not have a traditional computer science degree. But if you work through, say, the AWS Certified Solutions Architect Associate certification, you will get exposed to the core AWS services and architecture patterns that industry professionals use. You won't have to worry that you missed big topic areas in your job prep. The cert puts it all in one place. Very convenient. In this sense, *it almost doesn't matter whether or not you ever sit the cert exam!* Just studying the material gives you the real value.

Certs help you get interviews

When a recruiter is reading through your resume, a cloud certification does stand out. It says you're serious about the field and that there is some

external validation of that — especially if you are missing a traditional credential like a college degree in computer science. The cert's not going to get you the job, but it might get your foot in the door for the interview.

Certs are a big advantage … for one particular type of job

Professional services companies like consulting shops and MSPs are among the biggest employers of cloud-certified people. That's because certs are very valuable in the consulting world:

- Certification counts help the consulting companies maintain important partnerships with cloud providers like AWS and Microsoft. More people with certs add up to higher partnership tiers and better benefits.
- Certs help communicate expertise to the consultants' clients. If I'm hiring an outside consultant, I want to see credentials!
- They are required for certain types of consulting engagements (for example, AWS only allows certified Solutions Architect Professionals to conduct an official AWS Well-Architected Review).

Big consultancies have a bottomless appetite for cloud-certified professionals … assuming you actually know what the cert says you know, of course! So you really open up your job options by getting certified.

How many certs do I need before applying to cloud jobs? Is the CCP (or equivalent cert on another cloud) really enough?

I recommended the CCP in the original challenge because I like it as a formalized way to introduce beginners to cloud concepts - not so much because it is impressive to hiring managers. The CCP (or an equivalent cert such as Azure's AZ-900 or Google Cloud's Digital Leader) is really intended more for nontechnical roles.

So, here's a pro tip: I think that nearly **everyone I've seen get their first cloud job through the Cloud Resume Challenge had earned a more advanced certification**, such as one or more of the AWS associate-level technical certs. I've seen a few exceptions to this rule, but not many.

While in one sense, more certs are always better (I've never *not* been impressed by someone who holds all 12 AWS certifications — that's an impressive achievement by any definition), you do reach a point where you need to focus on building other skills. Otherwise, you run the risk of appearing like you are all head knowledge and

no practical ability. That's what the rest of the challenge is for.

The front end

Many people find that setting out basic HTML and CSS is the easiest part of the challenge for them, and so they speed through steps 2-6 only to find that the challenge gets much more difficult around step 7.

This is not because front-end development is easy. It's because the challenge has very limited, basic requirements for the static site. I did this by design, for a couple of reasons:

- I'm not a front-end developer, and I'm not qualified to set you a really cutting edge web development challenge!
- You are probably not going to be slinging a lot of front-end code in a cloud engineering role. You're going to be interviewing for infrastructure and ops jobs where your ability to center a div is not that relevant.

Creating the static site is more about completeness, and about having a project you can easily demo in an interview, than it is about building a web client that will really knock everyone's socks

off. (Though I've seen some really slick challenge sites out there!)

But that said, having to work with gluing a front end and a back end together is SUPER important. Part of what makes steps 7 and following such a challenge is that you're not just creating an API in a vacuum. You have to figure out how to communicate securely with a browser, handle client-generated errors, account for cache, and all sorts of other things that cloud engineers deal with every day.

All of which is to say: don't get complacent here! You are getting the pieces in place for the big challenges to follow.

The back end (the hard part!)

There are a lot of half-finished Cloud Resume Challenges out there that never get past steps 7-10. Because this is the part where you have to write legit code, in both Python and Javascript. If you have not written code in a programming language before (and, frankly, even if you have) it is painful, time-consuming, and open-ended … even though when you finally get done, you'll probably find that you've really not had to write that many lines of code at all.

So this is the part of the challenge that separates the wannabes from the champions. If you can build the back end of the Cloud Resume Challenge, you can do anything. Specifically: you can get hired as a cloud engineer. I'm quite confident of that.

OK, enough hype. It's a big chunk of work. Here's how I would break it down. You should be able to find online tutorials that cover each of these individual steps.

Goal 1: Get your Lambda function talking to the database (challenge step 8)

1. Set up a Python Lambda function using the AWS web console (don't worry about using SAM yet).
2. Set up a DynamoDB table, again using point and click.
3. Figure out how to write the Python code in your Lambda function that will save a value to the DynamoDB table. You'll just be testing your Lambda function from the console at this point. You'll need to figure out IAM permissions between the function and the table as well.

Goal 2: Trigger your Lambda function from the outside world (challenge step 9)

1. Put an API Gateway route in front of your Lambda function, still using the web console.

2. Make sure you can test the function from the API Gateway's internal test console and get some sort of "Hello World" response back from your code. You may need to workshop permissions a bit.
3. When this is working, grab the public URL for your gateway and test it using an external API testing service like Postman. You want to see the same output from your Lambda function there.

Goal 3: Trigger your Lambda function from your static site (challenge step 7)

1. You'll need to write a tiny bit of Javascript to make the HTTP request to the API Gateway.
2. Make sure you account for CORS issues between your browser and the API. This is almost certainly the most common and most frustrating issue faced by challengers. Good news: everybody eventually figures it out!
3. Verify that you can see output from your Lambda function returned from the Javascript call and printed out somewhere in the web app.

If you can get through these three chunks of work, you've solved the back end. (I know, I know. Easier said than done. If you're budgeting time for the challenge and you're new to cloud and code, expect to spend easily 60% of your overall time

commitment on these three chunks.) All you need to do now is automate!

Automation

I encouraged you to use the AWS web interface in the previous section because I think it's the best way to explore cloud services that are brand new to you. But as the challenge prompt states, this isn't the way professional cloud engineers deploy services. For that, you need infrastructure as code.

If I got my API working using the web GUI, do I really need to go back and write automation?

YES. Representing cloud resources as code using config tools like CloudFormation or Terraform is probably *the* most characteristic task in a cloud engineer's day-to-day job.

There are parts of this challenge you can hand-wave a little bit (nobody really cares if you kinda stole your static site template from somewhere else, for example).

But you *cannot* skip over the infrastructure automation piece. It is the most relevant job skill you will practice in this entire challenge.

The challenge recommends using AWS SAM as your IaC tool. Later on in the book I'll suggest some mods to the challenge, one of which is using Terraform instead. While SAM is an AWS-native option used by many teams, Terraform is far more widely adopted in the industry and it would be a great choice here.

Automating the deployment of your code through GitHub Actions is a fairly tutorial-driven task, so we won't spend much time on it here, but I do want to call out some complexity in step 11 (testing).

How to write your very own smoke test

Writing tests for your code is never a step you should skip, even if your code is simple enough that the tests feel perfunctory. But for the purposes of this challenge, there's a specific type of test you can write that should generate great conversation in a job interview. It's sometimes called an "end-to-end test" or a "smoke test". (Somewhere, Daniel Singletary is smiling!)

The original challenge prompt steers you toward writing "unit tests" using something like Python's built-in testing library. In a unit test, you verify that your code returns expected values before it is deployed.

This is fine, but not very interesting, since mostly what your code is doing is talking to cloud services. Without actual services to call, your code will have

to "mock", or pretend to call, those services - and that means you're not really testing much of anything.

On the contrary, an end-to-end or smoke test is run *after* your API gets deployed. It uses the real, live URL of your API and makes sure that it provides the expected responses to a test call. This indirectly verifies that

1. your Lambda code is working as expected,
2. permissions and other config are correct between your gateway, your function, and your database, and
3. all your resources have been deployed successfully.

An excellent tool for running smoke tests is called Cypress; it's a Javascript testing framework that (among other things) lets you call your API endpoint just like the Javascript code in your front-end app is doing. The Cypress test doesn't have to know anything about how your Lambda function is written, or what language it's written in, or even that it's a Lambda function. It just needs to know what output your URL should yield in response to input.

If your smoke test happens to fail, I would suggest adopting a "roll-forward" strategy to fix your deployment. Rather than trying to have your pipeline go back and deploy a known working previous version of your API, just push fixed code to your repository and let the pipeline run again.

Add a Cypress test as a GitHub Action after your deployment finishes, and you'll have an API you can feel confident is working … and a story you can confidently tell.

Writing a good blog

After weeks of work, multiple late nights, you've finally finished the entire Cloud Resume Challenge. It works from front to back. You've even passed a certification or two. You'd be perfectly happy never to think about any of this again. The last thing you want to do is sit down in front of the computer again and … write about it.

I know the feeling, believe me. (I've felt it plenty when writing this book!) I am imploring you, though, not to skip this step. It's kind of the punchline of the whole challenge. It's a signal to the world that you are participating in the cloud community, that you are giving back to future challengers, and that you are serious about leveling up your cloud career.

Recruiters may read it. Hiring managers may read it. Basically, the upside is unlimited and the downside is nil.

So just suck it up and write the dang blog, then share it on LinkedIn and Twitter. Worst case, you end up with good notes on what you built so that you can review them when prepping for interviews.

Realistically, you'll get some love from your network, and a few followers who are interested in what you build next - which is a great kickstart for a blogging habit that will grow your career over time.

Best case … well, best case, your blog goes viral like Daniel Singletary's. I'm not promising this will happen to you. But, having reviewed hundreds of challenger blogs, I can provide a few tips to help you stand out from the crowd.

Include a diagram
This is the #1 pro tip for a great blog, right here. People love pictures. Make a little architecture diagram of what you built in Lucidchart or draw.io. Share it as a standalone image on social media along with the link to the article. It will get engagement, guaranteed. (Here's an example of a challenge blog that makes great use of images.)

Make it about you, not about the challenge
The Cloud Resume Challenge itself has been explained and analyzed to death by the last six hundred bloggers. Your troubleshooting experiences are probably not unique. What is unique is *you.* Why are you undertaking the challenge? What unusual skills do you bring to the table?

Go back and review Daniel's Plumbers Guide to Cloud again. Notice how he reflects on his

plumbing skills and how they helped him learn to code. What life experiences do you have that helped you crack the challenge? Another challenge champion blog that I really like talks about the author's experience with mainframes, and how it felt to transition to serverless applications. That's an interesting story!

Framing your blog as a journey of personal discovery and growth, rather than a technical tutorial, will help you get more traction when you share it.

Don't ramble on
There's a famous line, "If I'd had more time, I'd have written less." Short and to the point wins in our hyperinflated attention economy. If your blog is more than a thousand words, you're probably rambling on. Save your in-depth observations on every error message you Googled over the past six weeks for your personal notes. Just focus on the takeaways that can benefit everyone.

Tag me
Seriously! I read and comment on just about every single Cloud Resume Challenge blog that I get tagged in on LinkedIn or Twitter (I'm @forrestbrazeal in both places). I'm always happy to help boost your reach.

Code review

Early versions of the Cloud Resume Challenge included a bold offer: I said I would provide personal code review, upon request, for anyone who completed the challenge. I figured maybe ten people would take me up on this.

You'd think somebody whose whole job involves preaching about "infinite scalability" would have been a little smarter.

Eventually I had to phase out that offer because, well, the challenge got a lot bigger than I anticipated. For awhile there in the summer of 2020 I was either working, sleeping, or reviewing pull requests. I saw YAML when I closed my eyes at night.

My personal nightmare aside, code review is still a really important step. Professional code isn't written in a vacuum. Team members look it over, provide feedback, suggestions, and tweaks, before it ever sees production. (That's the way it *should* work, anyway.)

So I highly recommend that you find an experienced developer in your life to review your challenge code before you start inviting hiring managers to look at it. If you don't have such a

person, try the Cloud Resume Challenge Discord; we have a "code-review" channel there where you can drop your work for feedback from a member of our community. (And hey, it might still be from me!)

In the meantime, I've collected some of the most common feedback I provided to hundreds of Cloud Resume Challengers on their back-end code repositories. (The front ends tended to be pretty wildly divergent, so you're on your own there.) When you finish your challenge, try running down this list and see if any of the suggestions apply to your API.

Lambda gotchas

Consider using structured JSON objects as logs -- this makes them easier to search and filter in log analysis platforms. (Very important if you have lots of users hitting your Lambdas!) Here is a great post from AWS Hero Yan Cui with some ideas on how to structure Lambda logs.

Don't hardcode your DynamoDB table name in the Lambda code. Instead, pass it as an environment variable or SSM parameter. (This will be helpful if you ever need to deploy multiple versions of your stack in the same environment.)

Consider initializing your boto resources outside the Lambda handler, in global space - thus, they will

only need to be initialized on cold start, which can speed up function performance.

DynamoDB gotchas
Many challengers introduce a subtle bug in their DynamoDB UpdateItem logic - what happens if the visitor count is not initialized (ie, on the first time the function is invoked?)

Are you making multiple DynamoDB calls in your function to get around the problem of initializing your visitor counter? You should be able to get this down to one DynamoDB call, which will also speed up your function performance. Hint: look into the semantics of the "ADD" operation on UpdateItem to make initializing values easier.

Don't give your DynamoDB table a custom name. This removes the ability to update some attributes without replacement in CloudFormation. Instead, let CloudFormation name the table, and pass the generated table name to your function as an environment variable or SSM parameter. (This is one of several CloudFormation best practices in this wonderful wiki article from AWS.)

Look into "on demand" billing mode for your DynamoDB table rather than provisioned capacity -- this will likely be a cost savings for a small site like this.

API Gateway gotchas
You can avoid having to define a separate API Gateway resource and swagger file by using an API "event" on your function.

Many challengers end up with IAM roles that are a bit permissive -- do you really need all those DynamoDB access rights? See how far you can scope that down.

Build and deployment gotchas
If you are writing unit tests, make sure you are importing the database update function from your main visitor counter file instead of redefining it in your test file. That way you can be sure you are actually testing the code that your application runs.

You should .gitignore your .aws-sam build directory -- no need to commit those files.

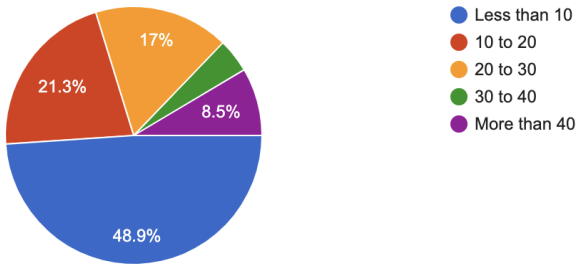
Planning your path through the challenge

How long will it take you to complete the Cloud Resume Challenge?

This really depends on your previous level of experience. In a recent community survey, I asked a bunch of challengers to self-report how many hours they spent working on the project. (Note that I didn't ask for an overall time window, so I don't

know if these people are working, say 8 hours a day for 5 days or 2 hours a night for 20. I also didn't ask them how long they spent preparing for their certification, since everybody's study habits are different.)

Number of hours to complete the challenge, as reported by participants



Huh! The divisions of this pie look pretty similar to the ones in the "previous experience" graph I showed you earlier, with years of experience mapping roughly onto faster challenge completion. From that similarity, we might draw the following weak rule of thumb:

Assume at least 40 hours to complete the challenge (not including cert prep time) if you have zero prior experience in cloud or software development. For each year of experience you have up to 3 years, shave ~10 hours off the expected time.

Your individual mileage will vary, and the amount of time you need is ultimately not important to the outcome. The challenge is not a race. Taking shortcuts (say, by peeking at other people's solutions online) just devalues your own learning experience. Focus on really understanding why you are completing each step, not on blazing through them.

With all of the above in mind, here is a sample mapping of tasks to time that you may find more helpful than the original 16-step order.

How I would approach the Cloud Resume Challenge *If you're not sure how you want to tackle the 16 steps of the challenge, try this method.*		
Projected Time Commitment: 50-70 hours over 6 weeks (adjust expected hours downward if you have previous experience)		
Week	**Challenge Milestone**	**Helpful Study Resources**
Week 1: Certification Prep (Challenge Step 1)	Complete a Certified Cloud Practitioner prep course and practice	A Cloud Guru Certified Cloud Practitioner prep course (this was free

	exam (10ish hours)	the last time I checked)
Week 2: Front End (Challenge Steps 2-6)	Get your resume laid out in HTML. Style it with CSS. Get the website deployed to S3 and fronted with DNS / CloudFront. (5-10ish hours)	CloudFlare (a CloudFront competitor) has an excellent series of articles explaining DNS and CDNs.
Week 3: Back-End API (Challenge Steps 8-10, 13)	Get source control set up. Get the visitor counter saving to the database. (10ish hours)	GitHub's Learning Lab is a good way to practice your git and hub skills. Get into all things AWS serverless with serverlessland.com.
Week 4: Front-End / Back-End Integration (Challenge Steps 7-9)	Get the visitor counter displaying on the homepage. (10-15ish hours)	There are a billion Javascript resources out there. I recommend

		going through a solid, concise tutorial course such as freeCodeCamp's before getting too far in.
Week 5: Automation / CI (Challenge Steps 12, 14, 15)	Get your infrastructure automated with SAM and deploying via GitHub Actions. (10ish hours)	Anything by Eric Johnson on SAM is typically excellent - start here.
Week 6: Tidying Up (Challenge Steps 1, 11, 16)	Add tests for your code. Sit and pass CCP exam. Write blog post. Celebrate! (as much time as you want/need)	If you want to get better at writing technical blogs that people actually want to read, Philip Kiely's book "Writing for Software Developers" is certainly worth the investment.

The 17th step: Making the challenge your own

If you Google the Cloud Resume Challenge today, you will find pages (and pages, and pages… and pages) of projects that people have completed over the past 18 months. Most of these projects look very, very similar - in fact, I'd say a good 50% of them use one of the same two or three common HTML resume templates.

I'm not accusing anyone of cheating. I'm just pointing out that as time goes by, the marginal value of completing this project does in fact decrease. There are just so many competing projects out there, all following the exact same 16 steps, that each individual accomplishment becomes slightly less impressive.

(This is a common phenomenon noted in college courses, by the way. If professors assign the same coding homework year after year, student scores invariably inflate - not because incoming freshmen are getting smarter, but because the corpus of solutions floating around out there from past students gets bigger and easier to crib from every year.)

But take heart! The Cloud Resume Challenge is not so easily dismissed. You just need to mix in a bit of creativity.

So I hereby add a 17th step to the challenge, just for you brilliant handbook readers. Here is the 17th step:

17. Spice it up
Tweak one or more of the previous steps to make the challenge your own. This will help you stand out and give you a better story to tell about the challenge in your job interviews.

I highly recommend - nay, I adjure you - to come up with some kind of a unique spin on the challenge, some technical twist that requires your own creativity and insight. It will make your project, your code repository, your blog post stand out.

Now, this twist might indeed become the most important talking point for you in a job interview, but make sure it's a choice *you can defend.* For example, it would be theoretically possible to serve your cloud resume from a fleet of Raspberry Pis managed via AWS Greengrass … but *why would you do that?*

The goal here is to build something that makes a hiring manager go "oh, that's relevant to the work we are doing here on Cloud Team X", not "that's

cool, but I would fire someone who built it on my dime." And if you're not sure where that dividing line is, it's best to stick close to the outline of the original challenge, which is pretty tried and true.

Just to get your creative juices flowing, here are some sample ideas for spicing up the challenge (all tried by previous challengers … but not by too many):

Now IAC You, Now I Don't
Replace SAM with a different popular Infrastructure as Code framework such as Terraform, Amplify, or the AWS CDK. Let the config wars rage in your soul!

When Will IAC You Again
The challenge prompt asks you to use infrastructure as code to define your back-end resources, but does not explicitly require this for your front-end resources (the S3 bucket, the CloudFront distribution, and the Route53 records.) Try to define these in something like CloudFormation as well, deploying them automatically through your front-end CI/CD pipeline.

Kiss Me K8s
Instead of Lambda, serve your back end API on a different cloud-native technology. Kubernetes would

be cool - just make sure you can defend the complexity of your setup!

No REST for The Weary
Instead of API Gateway, try using a GraphQL gateway such as AWS AppSync. I actually don't know if I've seen someone do this, but it would be an *excellent* project mod because GraphQL is so hot right now. It would also pair nicely with Amplify as your IAC tool.

Branching Out
Instead of GitHub Actions, try using a different CI/CD tool for building and running your tests and deployments.

AWS's native CI/CD tooling is called "CodeSuite" and includes CodeBuild (a Docker container that runs your tests and, well builds your code) and CodePipeline (orchestrates your deployments). These tools are commonly used in AWS-heavy cloud shops and will look good on your resume.

You could also consider another third-party tool such as GitLab, but be aware that you may run into charges.

Foreign Relations
Run a relational database instead of DynamoDB, maybe Postgres or MySQL. You'll have to defend this because it will be less "Serverless" than the

rest of your setup, but it will get you into some challenges commonly faced by real cloud teams, particularly around networking. (How do you connect a Lambda function, which is stateless, to an inherently stateful database?)

Monitor Lizard
Add some monitoring to your setup - make CloudWatch alert you if your Lambda usage goes over a particular threshold, say. Be sure to automate the creation of any and all monitoring resources. Watch for billing on CloudWatch, though, it's sneaky expensive.

On Again, Auth Again
Improve the authentication and rate limiting between your front end website and your API Gateway. The original challenge prompt leaves this murky, and many people ship an API with zero protections. (A good hiring manager may well ask you about your approach here, so it's in your best interest to be proactive with this one.)

Framework Makes the Dream Work
Instead of vanilla Javascript, use a modern web framework such as React to define and generate your front end. This will necessitate some changes to your build automation as well. Good mod if you really want to sell this project as a "full stack" project in interviews.

Cert Hard 2: Cert Harder
I've mentioned this earlier when talking about certification, but it's a really good idea to have a more advanced cert on your resume than just the AWS Certified Cloud Practitioner or equivalent. While the AWS associate-level certs are all great, you might find the AWS Networking Specialty cert or AZ-700 to be just as helpful in prepping for interviews, which tend to have a heavy network component. Be warned, though: those are tough exams!

Set The Stage
Create one or more separate cloud environments where you deploy your code for testing before it is finally deployed to the official "production" environment. You will need a multi-stage pipeline and tests for each environment. You will also need a plan for how to structure your source control lifecycle (look into "Git flow"). Hiring managers will love this. Good luck!

For even more ideas, including porting the entire challenge to other cloud providers, check out the "Helpful resources" section at the end of this chapter.

Stephanie's Story: From Bank Manager to Cloud Engineer

"I took on the Cloud Resume Challenge because I wanted to gain real life experience in a brand new field. The challenge tested me on whether a career change was really what I wanted, as well as how quickly I could learn something new.

"The challenge took me about 150 hours to complete over about 6 weeks (though I really started hitting it hard in the last 3 weeks or so). It took me longer in part because **I chose to make the challenge my own by doing my automation in Terraform instead of SAM**.

"As it turned out, this was one of my best decisions! I was able to speak to real life scenarios based on what I had faced during the process. One that stands out the most was a troubleshooting exercise I was given during an interview. I was able to answer quickly and confidently because it was something I had run into during my own resume challenge.

"After a 3-month job search, I am now an Associate Cloud Engineer for iRobot, and I freaking love it! I learn something new every day. Love my team, love the work and enjoy challenging myself every day."

Helpful resources

The double-edged advantage/disadvantage of starting the Cloud Resume Challenge now, as opposed to 16 months ago, is that the challenger community has created a *ton* of support resources to help you.

Why is this a disadvantage? Remember, the burning pain of sitting up late with fifty tabs open, trying to figure out why your Lambda function won't return anything, is actually the most powerful part of the learning experience. And I still recommend that you avoid looking at anyone's actual solution code if you can help it.

Think of other challenge solutions as spoilers. You didn't want to know the ending of Avengers: Endgame before you entered the theater. Don't let some other developer spoil the learning experience and pride of crushing the challenge on your own merits.

But with all of that said, there are still some wonderful helps and extensions to the challenge out there for anyone who's curious, masochistic, or just plain stuck. Caveat: I don't own most of these links, so apologies in advance if they go dead for any reason.

The Cloud Resume Challenge Discord Server
This is the place to be if you are working on the challenge. Join 1500+ other cloud enthusiasts ranging from beginners to experienced mentors. Ask your challenge questions, get code review, view cloud job postings, or just chat about cloud and life. It's free and very chill. I'm there. What more reason do you need? Join us already.

The Cloud Resume Challenge Video Tutorials
Lou Bichard, founder of "Open Up The Cloud" and tireless friend of the challenge, has put together a comprehensive video series walking through each step of the original challenge. If you're a visual learner, or are having trouble figuring out a specific step, Lou's videos are a fantastic resource. His associated GitHub repository also curates a nice selection of blogs written by challenge champions.

The Challenge on Azure
One of the most common questions I get is "can I do the Cloud Resume Challenge on Azure services?" The answer has always been YES, but until recently there wasn't a great guide to follow. Happily, A Cloud Guru has blessed us with an all-Azure version of the challenge presented by Microsoft cloud advocate Gwyneth Peña-Siguenza. You can run through her text prompt or watch her thorough video on the subject (requires free account).

The Challenge on GCP
Yeah, one of the *other* most common questions I get is "can I do the Cloud Resume Challenge on Google Cloud services?" Good news, A Cloud Guru has stepped into the breach again, this time with an all-GCP spin on the challenge presented by Mattias Andersson. (Sorry, Oracle Cloud - I've no bias against making an OCI version, but not one single person has asked yet.)

The Challenge in Spanish
Challenger Jesus Rodriguez generously gave his time to translate the original challenge, as well as some FAQ, into Spanish. If you'd like the Cloud Resume Challenge translated into another language and have the know-how to do so, please reach out - we'll figure out a way to make it freely available on the challenge homepage.

The Best Jobs in Cloud
My regular "Best Jobs in Cloud" newsletter rounds up job opportunities from people I know and trust in the cloud community. The best part: each job comes with direct contact information to reach out to an internal referral inside the company, no blind applications required.

The job roles range from junior to senior, and many of these hiring managers are familiar with the Cloud Resume Challenge and love to hire champions. If

you've bought this book, you're already subscribed to The Best Jobs in Cloud at no extra charge. If not, you can sign up here.

Part 2: Extra-Challenging Extra Challenges

While the Cloud Resume Challenge covers a lot of ground in just sixteen steps, it is not (and I know you're surprised to hear this) a one-stop shop for picking up every single skill a cloud engineer needs to know.

However, the challenge format itself - take a project spec, go down the rabbit hole, emerge with skills and stories you will never forget - is pretty extensible, and so almost since Day 1 resume champions have been clamoring for more challenges on other cloud and DevOps topics: Kubernetes, event-driven programming, and so forth.

In this section of the book, I've rounded up several existing challenges created in a similar spirit to the original CRC, as well as four brand-new challenges contributed by our challenger community. Happy building!

The Cloud Guru Challenges

I've already mentioned the Azure and GCP "Cloud Guru Challenges" created under my supervision at A Cloud Guru. These are part of a larger attempt to extend the general challenge format to a variety of technical projects, and the response has mostly been pretty good. Note that each of these projects includes "extra credit" ideas for making it your own - take advantage of that, as most folks don't!

Event-Driven Python on AWS
I wrote this project myself, and so I'm glad that it seems to have helped a lot of people. It takes really core, bread-and-butter enterprise cloud skills (working with data warehouses, ELT pipelines, Python, and events), mixes them with some timely COVID-19 data sets, and walks you through building a real-time dashboard that's sure to start a conversation. A great project to take into a data or DevOps engineering interview.

Machine Learning on AWS
Build a movie recommendation engine with AWS ML Hero Kesha Williams. This one is *tough*. It's one of the rare cases where you'll gain plenty of learning even if you start by reading her follow-up

blog where she explains exactly how she did it. Great for data scientists and data engineers.

Multi-Cloud Madness
OK, so this one's a little contrived, but still a fun way to figure out how to get integrations working between multiple clouds. Do Scott Pletcher's challenge if you have an interview somewhere that's wrangling AWS and Azure in the same shop.

Building a continuously integrated global web app on Azure
Lars Klint's challenge will seem fairly easy if you have Azure experience, but is a great way to transfer knowledge from other clouds. Also a good way to get CI/CD reps, as it incorporates Azure Pipelines pretty heavily. Good choice for a cloud or DevOps engineer interview at an Azure shop.

Improving application performance on AWS
Here's where you can get your hands cloudy with some good old-fashioned servers. Stand up a web app, figure out how to make it faster with judicious use of caching. A great project to take into a cloud architect interview.

Jakob's Story: From Clinical Research Technician to GCP Engineer

Time to complete the Cloud Resume Challenge: ~100 hours, including CCP study

“The Cloud Resume Challenge helped me a great deal in my job search because it gave me something to talk about and refer back to.

“By far the most useful thing was my understanding and implementation of infrastructure as code. Setting up pipelines to test and deploy my changes seemed like an unnecessary step at first, but I think that being able to discuss how it made quick and safe iterations possible made me a much more attractive hire.

"Finishing the Cloud Resume Challenge showed me that a normal person like me could build things accessible to the whole world on the cloud. It changed my inner monologue from 'I wish someone would build something to address this need I have' to 'I could build something to address this need I have.'

"The Cloud Resume Challenge is a great way to open your eyes to your own potential. After you finish, KEEP them open and build the solutions you want to see in the world. People will notice that."

In addition to the resources above, I'm honored to feature four brand-new community challenges in this book. Daniel Dersch has hired numerous resume champions onto his GCP-focused cloud team. Justin Wheeler is an inveterate challenger who holds the distinction of being the only person (so far as I'm aware) to complete every single Cloud Guru Challenge. Jennine Townsend is one of the most deeply knowledgeable AWS infrastructure experts I know.

I asked each of them to create a challenge that would impress them when talking to a job candidate. And they delivered. Daniel's Kubernetes and open-source challenge, Jennine's two AWS developer-focused challenges, and Justin's meta-resume challenge will all help you solidify your project list and start conversations hiring managers love.

Building an automated, disposable AWS development environment

by Jennine Townsend

In this challenge, you will build a simple template to start an AWS Cloud9 environment, along with a CodeCommit repository holding setup scripts.You will be able to quickly initialize a fresh useful isolated development system with fast networking and whatever CPU and RAM you need whenever you need it, and throw it away without fear when you are finished with it.

AWS's Cloud9 is a cloud-hosted development environment. You might prefer another IDE, or often use Docker or a virtual machine for isolated environments, but sometimes it's handy to be able to quickly spin up a whole isolated environment in the cloud, with the advantage that you can just throw it away and spin up a fresh one whenever you like!

Prerequisites and helpful resources

Some familiarity with CloudFormation or a similar tool would be helpful. Note that this is quite a simple and self-contained template, so this project in itself could be helpful in learning your chosen tool.

- Cloud9 documentation: https://aws.amazon.com/cloud9
- “How does AWS Cloud9 work?”, including a nice diagram:
- https://docs.aws.amazon.com/cloud9/latest/user-guide/welcome.html#how-does-it-work
- Jared Short's blog post: https://www.trek10.com/blog/i-buy-a-new-work-machine-everyday
- Documentation for the CloudFormation resource:
- https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-cloud9-environmentec2.html
- Here’s a recent post on the AWS Architecture Blog that might give you some ideas: https://aws.amazon.com/blogs/architecture/field-notes-use-aws-cloud9-to-power-your-visual-studio-code-ide/

Challenge steps

1. Cloud9 initial exploration

If you have never tried Cloud9, go ahead and click through the console setup for an "EC2" environment, and get a feel for what it does. Notice how the system will “stop” after being idle for a

while (so you're not paying for CPU+RAM while it's stopped), and then auto-starts when you resume work.

Read Jared Short's blog post, and try to absorb the view of Cloud9 environments as temporary and easily replaced.

Don't forget to destroy the environment (and all the resources you use) when you're done! The console is convenient, but gets tiresome if you recreate an environment frequently, so we'll quickly move on to better automation.

2. Simple template

The CloudFormation resource AWS::Cloud9::EnvironmentEC2 resource is pretty simple. You can start out really just specifying the instance type, so do that!

There's a fairly complicated and slightly out of date template in https://github.com/aws-samples/aws-cloud9-bootstrapping-example/, but you can start out with a very minimal one. Make sure you check the CloudFormation documentation for AWS::Cloud9::EnvironmentEC2 for the current available parameters.

Then look at the other things you can set, such as what OS to run, and what CodeCommit git repositories to clone.

Try out your template, create and destroy a few environments, and try out a couple of the OSes.

3. Add a setup script
You might have a script you run on a new system, to set up your prompt and install your preferred tools and so forth. Try running that on Cloud9, and tweak it to work well there.

Jared Short mentioned Ansible in his blog post; Ansible is a configuration management tool, so it is very well-suited to this kind of thing. My setup script is a shell script that installs or upgrades packages on the basis of environment variables, and it configures git, sets up convenient aliases and shell completions, and even grows the disk if I choose!

4. Have Cloud9 check out your script at startup
Recall that an environment can check out a CodeCommit git repository upon startup so it's already there when you first log in. Now have a look at the CodeCommit console, and create a repository there (I call mine "cloud9-cfg").

Depending on your familiarity with git, you could check in your script, or just upload it to the repository from the console.

Now add that repository to your template, and next time you start an environment, your script will already be available on the instance! You might want to add more repositories to the list (I check out my personal FAQs repository in every environment, for example).

5. Now think about what you might use this for!

- As you work on a project, you might want to add the project repository to the list in the template, or check it out in the setup script.
- Have you wanted to try out a new version of a language, but don't want to risk colliding libraries on your own system? Make another script in your setup repository that installs the language tools of the specific version you want to try out.
- Are you writing code that does a lot of network calls while you're developing and testing it? It's convenient to have an environment in the cloud, where the network is very fast.
- Are you trying out something new, that's changing rapidly? While AWS CDK was in constant flux, I tried it out using Cloud 9, and just constantly updated the version that my script would install.

- Do you want to collaborate with others? Cloud9 includes collaboration capabilities.
- Do you *sometimes* need more power? You can even launch an environment with 32 CPUs and 128 GiB of RAM for less than $5 for three hours -- but be extra sure you don't leave **that** running for long!
- If you'll be teaching a workshop, consider using Cloud9 and providing setup scripts to save time spent helping each individual configure their laptop -- all they need is a web browser and an AWS account.

6. [optional] Understand and tweak permissions
The user account on Cloud9 has your IAM permissions, in case you want to connect to other AWS services. You can attach an IAM role / instance profile, and configure Cloud9 to use that instead, if you prefer (for example to test code using the IAM role that its destination instance or service will have). Read the "Security" chapter in the Cloud9 User Guide, and particularly the "Identity and access management" section for details.

7. [optional] Tweak networking
Read the Cloud9 documentation about networking. Try inbound and outbound connections, and understand the security implications of opening up an inbound connection.

8. Consider security
Think about your attack surface. What is exposed? Does your development process involve web servers or other processes that listen on the network? Are they reachable from the Internet? What data do you have on the system? Does your setup script download and run code from elsewhere? Do you trust that code? Does your AWS account have permissions to affect other systems? Test and document all this.

9. Write a blog post!
Re-read Jared Short's blog post. Now that you have this new tool available, how do you think you might use it? As your main development environment for day-to-day work? As an easily-disposable, but capable, blank-slate test environment? Will you use the collaboration possibilities? Would you like to try out an exotic new language?

Extra credit

Here are several ideas you can use to spice up your environment or make the automation more flexible.

One-click launch
Sometimes you'll see a "Launch stack" button especially on an AWS documentation site like this: https://docs.aws.amazon.com/AWSCloudFormation

/latest/UserGuide/sample-templates-applications-us-west-2.html

Figure out how that works! Set up your template with a form where you can specify the instance type and (via a Mapping) what OS to use. This makes it easier for you to show off your template to others: make sure that the "Launch stack" button works for others, setting up an environment in their account.

URL output
Add a stack output that constructs the console URL to connect to the Cloud9 environment that the stack built, so you can quickly point your browser at it as soon as the stack completes, maybe even scripting the process of retrieving the URL and sending it to your browser.

Persistent storage
Do you think you might want permanent storage that doesn't go away each time you replace the environment? Look into S3 and EFS, or you might attach an EBS volume with a filesystem on it. Maybe you could keep data on S3, and copy it to the system's local (temporary) storage in your startup script.

Systems Manager
Understand the difference between "EC2" and "SSH" Cloud9 environments -- see the documentation to learn more.

Instead of using a setup script, maybe you can use an AWS Systems Manager Run Command document to perform some setup steps.

Elastic volumizer
Have your script grow the root volume.

Some hints: I get the instance ID using curl from instance meta-data, from that I use the AWS CLI to derive the root volume ID, and then use the CLI to modify the volume. After a few seconds' sleep just in case, then it uses the Linux "growpart" utility to tell the system to notice the added space, and "resize2fs" to grow the filesystem to fill the space.

Figure out how to do this, and don't worry about damaging the root filesystem -- if you make a mistake, you can just delete the CloudFormation stack (which terminates the system) and start over. Realize that the root volume goes away with the system, and is not permanent storage.

Final takeaways

By now you realize that this challenge is really about two things: building familiarity with a genuinely useful AWS offering, and learning how to use CloudFormation to make creating and deleting cloud resources quick and easy. But there's also an underlying purpose: solidifying the idea that

cloud resources, even ones that feel as personal and customized as a development environment, **should** come and go quickly and easily.

Working with data on Kubernetes

by Daniel Dersch

Goal

Deploy a workload to Kubernetes to pull public data and import it into Elasticsearch or Prometheus for analysis. Completing this challenge means that you can work with Docker, Kubernetes, Helm, Prometheus or Elasticsearch, Grafana or Kibana, and a programming language.

Challenge steps

1. Install local Kubernetes

1. Install Kubernetes to your local machine. There are various options for accomplishing this; pick whichever one you want!
2. Install these tools for working with Kubernetes.

2. Choose a public time series data set

1. Browse data.gov or a similar site for a time series data set that you are interested in.

2. The data set must be active and changing. In other words, there should be additional data in the future.

3. Install either Elasticsearch and Kibana OR Prometheus and Grafana

1. Elasticsearch is an extremely popular search and analytics engine, and Kibana is the Web UI for working with data in Elasticsearch.
2. Prometheus is an extremely popular time series database, and Grafana is a Web UI for working with data in Prometheus.
3. You'll be storing your public data in either Elasticsearch or Prometheus. The choice is up to you, so research both tools to determine which one is better for your data!
4. Use a tool like Helm to install your chosen tool to your local Kubernetes.
5. Make sure you can login to Kibana (if you chose Elasticsearch) or Grafana (if you chose Prometheus) to complete this step.

4. Write your importer program

1. Write a program that pulls your public time series data set from step 2 and imports it into Elasticsearch or Prometheus.
2. Use any programming language or shell you like for this step! Just make sure whatever

you use has a good Elasticsearch or Prometheus library SDK.
3. Your program will likely need to transform the data into a structure compatible with either Elasticsearch or Prometheus.
4. The program must accept the URL of the public data set as a parameter.
5. You will likely have to parameterize this URL in your code with query strings.

5. Write your Docker image

1. Write a Docker image that contains your program from step 4.
2. The Docker image must accept an environment variable named URL or URL_TEMPLATE that gets passed to your program.

6. Run your Docker image as a container on Kubernetes

Run your docker image as a workload on Kubernetes. The workload should automatically run once per day every day to import new data.

7. Visualize your data

1. Connect to Kibana Grafana and create a chart comparing time to some attribute of your data.

2. For instance, if your data was COVID-19 cases worldwide, you could create a chart with "Day" on the x-axis and "Count" on the y-axis.

8. Create an actual Kubernetes cluster using a managed service in a public cloud

At the time of this writing, I recommend a Google Kubernetes Engine (GKE) cluster in a single zone, which should have minimal charges.

9. Run your workload on your new cluster!

1. Install Elasticsearch/Kibana or Prometheus/Grafana to your cluster using Helm or a similar tool.
2. Run your Docker image as a container on the deployed Kubernetes.
3. Create your visualization on the cluster.

10. Teardown

Don’t forget to tear down your cloud resources when you are finished to avoid incurring ongoing expenses.

11. Write a blog post detailing your work!

Extra credit

Deploy the other tool!

If you initially chose to deploy Prometheus, then deploy Elasticsearch this time - or vice versa. Then, either:

- Configure your Kubernetes cluster such that all metrics are sent to Prometheus, or
- Configure your Kubernetes cluster such that all logs are sent to Elasticsearch.

Final takeaways

Kubernetes is the leading container-orchestration system, and Docker is the leading tool for building container images to run on Kubernetes. Elasticsearch and Kibana are key components of the ELK stack which thousands of organizations use for log analysis. Prometheus and Grafana are the most popular open-source tools for monitoring workloads on Kubernetes and other platforms. Every engineer on a DevOps team should be familiar with deploying and working with these tools.

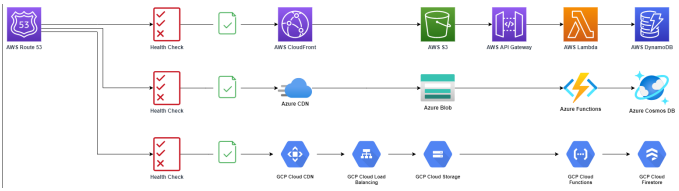
Build a multi-cloud resume (the "Meta-Resume Challenge")

by Justin Wheeler

In this challenge you will be asked to build a traditional three-tier application that will span three unique cloud providers. The application itself will be your virtual resume that can act as living proof of your cloud expertise.

The frontend tier will be constructed with web technologies like HTML, CSS, and JavaScript. The backend tier will be constructed with server-side technologies like Java or Python. Finally, the database tier will store the application data.

The services used will be serverless. The resources will be managed using Infrastructure as Code (IaC). The code will be deployed automatically using a Continuous Integration (CI) / Continuous Deployment (CD) pipeline.



While you would rarely if ever build a real-life application that is redundant across multiple cloud providers in this way, doing so as an exercise provides several advantages:

- Improved cloud knowledge by using and learning the subtle differences between providers.
- Improved options for cost savings since one provider may offer a similar service for a better price.
- Improved product offering since you can choose from the range of products from all providers.

Prerequisites

Review these resources before getting started:

- DevOps Essentials: https://acloudguru.com/course/devops-essentials
- IaC Pulumi: https://www.pulumi.com/docs/get-started/
- IaC Terraform: https://learn.hashicorp.com/collections/terraform/aws-get-started
- Serverless Concepts: https://acloudguru.com/course/serverless-concepts

- Serverless Resources AWS: https://aws.amazon.com/serverless/
- Serverless Resources Azure: https://azure.microsoft.com/en-us/solutions/serverless
- Serverless Resources GCP: https://cloud.google.com/serverless

Challenge steps

1. Create a plan of what services and tools you will use

1. Which three cloud providers?
2. Which cloud services?
    a. All the services should be considered serverless.
    b. Sometimes there is some debate on whether a service is truly serverless. In these instances, I would argue to side with the cloud provider that owns that service.
3. Which DNS provider?
4. Which DNS registrar?
5. Which CI/CD service? I recommend GitHub Actions.
6. Which IaC provider? I recommend Terraform.
7. Which Git provider? I recommend GitHub.
8. Which programming languages?

9. Which programming frameworks?

2. Create a Git repository to store your project resources

All code should be stored in version control. CI/CD and IaC templates should be stored in version control.

3. Create IaC templates to create cloud infrastructure

There are multiple ways to manage templates. I recommend at least three templates separated by provider to keep the IaC templates from becoming cumbersome.

4. Create resume data in databases

a. Resume data should be hard coded on the website.
b. You could create your data in JSON documents so that they are easily uploaded to the databases.

5. Create front end code to display the website.

a. There should be some indication of which cloud is serving the application visible on the front end.
b. There are countless ways to build websites. You can keep it simple with HTML, CSS,

and JavaScript, or go more advanced with a front end framework like Angular, React, or Vue.

c. Ensure that your front end code is considered “static” so that it can easily be hosted on static website hosting solutions like AWS S3. Dynamic front end code like PHP or JSP will not work.

6. Create back end code to fetch resume data from the databases

Ensure that your back end code is capable of being hosted on your cloud provider services. For example, AWS Lambda only supports C#, Go, Java, Node.js, PowerShell, Python, and Ruby. So if you’re trying to use AWS Lambda with a different language, you may struggle.

7. Create CI/CD templates to orchestrate code deployment to the various cloud providers

a. There are multiple ways to manage templates. I recommend at least two templates separated by frontend and backend code.
b. Your CI/CD pipeline should have at least build, test, and deploy stages.

8. Register a domain name for your website

You can register a .com domain name for $12/year. Some cloud providers are also DNS registrars like AWS and GCP, which may simplify things if you choose to use one of these providers.

9. Configure DNS routing for your website

1. There should be a single DNS provider that controls routing to your domain name.
2. Configure health checks for your three endpoints so that you don't send traffic to an endpoint that is unhealthy.
3. Configure round-robin routing between the three cloud providers so that when users enter your domain name, they will be navigated to any of the healthy endpoints.

10. Write a blog post explaining your project

a. What did you learn?
b. Why did you make the decisions that you did?
c. What would you do differently next time?

Extra credit

Here are a few additional steps you can take to further improve your challenge.

Use less popular cloud providers

AWS, Azure, and GCP are very popular cloud providers with tons of documentation, examples, and tutorials. It would be more impressive if you were able to accomplish the challenge using a different cloud provider like Alibaba, IBM, OCI, etc.

Configure zone-apex routing
Zone-apex routing refers to your domain name without any prefix. Example given, “example.com”. This will take some additional configuration and may act different than your standard DNS routing. The reason is that it’s only available on some DNS providers and only with their services.

Configure proper security and authentication between application layers
The database should only be accessible by the back end; the back end should only be accessible by the front end.

Use a website testing tool like WebPageTest to ensure your resume is performing well
WebPageTest ensures your website is secure and efficient.

Final takeaways

This project tackles many prevalent concerns in enterprise companies.

- Multi-cloud: companies will typically employ more than one cloud provider.
- Fault Tolerant: companies will often worry about what would happen if a given cloud provider fails completely.
- Serverless: companies will look for serverless solutions to reduce expenses and improve agility.
- Infrastructure as Code: companies may prevent or prohibit access to the cloud consoles in production environments for good reason.
- Continuous Integration / Continuous Deployment: companies hate manual processes especially around updating code.

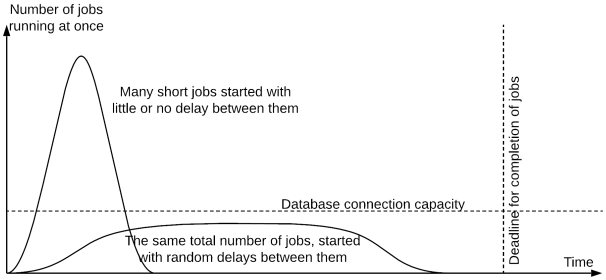
If you are familiar with any of the topics above, then you will have an edge over other cloud experts. If you complete this challenge, you will familiarize yourself with all these topics. So, what are you waiting for? The world needs more cloud experts! Good luck!

"Flattening the curve": Architecting asynchronous processes on AWS

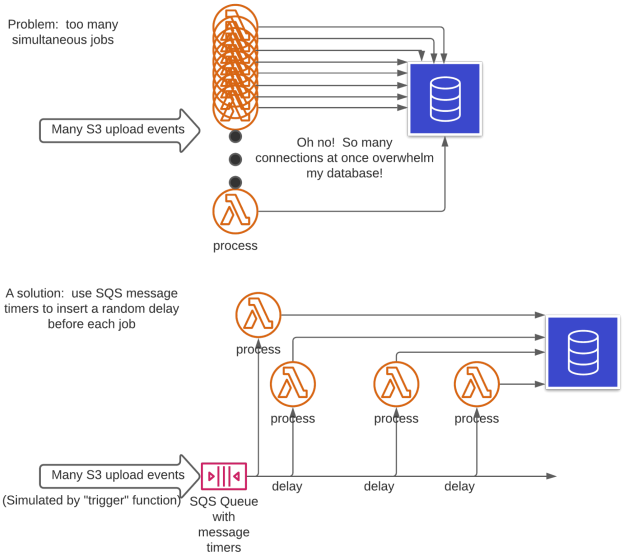
by Jennine Townsend

Goal

In this challenge, you will build a mechanism for triggering a burst of Lambda functions with some jitter in their start times so they aren't all running simultaneously.



You will understand a simple but reliable method to introduce jitter into a burst of cloud events, so that you can control their peak impact on downstream resources such as database connections or service limits such as Lambda concurrency.



Prerequisites

You will need just a little familiarity with AWS Lambda and the Lambda-supported programming language of your choice.

You will need to deploy a simple architecture; I suggest using CloudFormation or some similar tool, but you could build this in the console. You will also need to install the AWS CLI.

Challenge steps

1. Writing the "process" function

Recall the notion of “flattening the curve”: if we have a resource that we don’t want to overwhelm, we will want to spread out calls on that resource to remain below its capacity.

Write a simple Lambda function called “process” that just pretty-prints the “event” structure it receives to the logs. Learn how to find the logs in CloudWatch Logs and learn how to see when it ran, and how long it took, and experiment with running it several times quickly. (Hint: study the AWS CLI command `aws lambda invoke --help`.)

See if you can run the function a dozen or many more times as quickly as possible. This Lambda function will be a stand-in for one that reads and processes PDF documents and writes information to a database.

2. Simulate processing time

Update your “process” Lambda function to “sleep” for a few seconds, to simulate processing time. Trigger a few dozen of these as quickly as you can, including an event with a “document” key, and a different value for every event -- for example, you could use a shell loop around `aws lambda invoke …`

But in a cloud context, the peak can be even steeper! Imagine that we offer a service to customers who upload PDF documents to us, and

our “process” function summarizes the documents and inserts some information about each document into a database -- this is a pretty standard serverless use case.

But if a customer uploaded a hundred PDF documents at once, and all one hundred Lambda functions try to write to the database at once, we could cause a brief but very serious overload on the database! This is a common concern, and one with many possible solutions: for some ideas, read up on database connection pools and message queues.

3. Monitor initial function behavior
Using the Lambda console’s “Monitor” tab for your “process” function, look at its graphs. Notice the height and duration of peaks, especially the “Invocations” and “Concurrent executions” graphs.

Experiment with changing how long the function sleeps (takes to process documents), and how rapidly you invoke a burst of them. See how the peaks change if you put even a brief pause between invocations, or if the function sleeps (processes) for different amounts of time.

4. Create SQS queue
For this challenge, we will use a queue, and specifically Amazon’s Simple Queue Service (SQS). Read about SQS, and learn how to create

a queue and give Lambda functions permission to write messages to and read messages from it (your deployment tool, eg SAM or Serverless.com for example, might help with this).

Create a queue, and write a Lambda function called “trigger” which sends five messages to it which have a ‘{ “document”: 1234 }’ shape of JSON event structure.

Set the “document” value to a counter, for example, so you can tell your five messages from each other (`{ "document": 1 }` in the first one, `{ "document": 2 }` in the second one, and so forth).

Find the queue in the SQS console, and work out how to receive and view messages.

5. Trigger Lambda function from queue
Set up the SQS queue as a trigger for the “process” function. Once the “process” function reads and logs messages from the queue, look at the logs and the SQS documentation to see how the `{ "document": 1234 }` structure appears in its logs. You will see a “Records” array since SQS can send multiple records to each Lambda invocation.

Decide how to handle this; in my use case, which is somewhat different, I limit the “process” Lambda function to receive an SQS batch size of 1, but you

might decide to have your “process” Lambda function loop over “Records” and process each item. If you use the same database connection throughout, this can also help reduce the number of simultaneous connections.

6. Experiment with the “trigger” function
Have it send five messages, ten, fifty, and see what the “process” function’s logs and graphs look like. See how tall the peaks get! Think about what that many “Concurrent executions” might mean for your database (or API limits, or licenses, or other resources).

7. Incorporate message timer
Now read about SQS message timers. This is a mechanism for sending a message, but asking SQS to delay delivering it just a bit. This sounds helpful!

Update your “trigger” function, adding a message timer to delay delivery by a random number of seconds between 10 and 100, say. See how this affects the graphs for the “process” function. Compare the height of the peaks with and without the delay.

There’s a cost to this method: you don’t finish the work for your customer for up to 100 extra seconds, almost two minutes. Is this OK for your use case? Will your customers mind? This is a business

decision, and it's important to include in the tradeoffs.

If the customer has a dashboard which shows progress, and (because we're not overloading the database) that dashboard is fast and accurate, will that make them happy? See if they love an accurate animated progress bar.

8. Automate monitoring
This is a very simple method of "flattening the curve", and once set up it is very reliable. But things change, and your customers may be uploading ever more documents. Or you may find that your delays are longer than you need, that your database has plenty of capacity and you could shorten the delays.

Using CloudWatch Metrics, set up alerts on the "Concurrent executions" metric that will warn you if the figure rises above 10, say, and then tweak the values in the "trigger" function to test the alert. While you're there, set up alerts for "Error count", "Success rate", and other metrics you think need alerts.

Extra credit

There are many other considerations when connecting cloud services in a robust and scalable

way. Here are just a couple more rabbit holes you can explore using the infrastructure we've already set up.

DLQs
Read about Dead Letter Queues. Both Lambda and SQS can use Dead Letter Queues. Modify your “process” Lambda function to “fail” 10% of the time. Trigger it a few dozen times, and see what shows up in the DLQ. How might you automate monitoring a DLQ? How would you re-process items that show up there?

Delay optimization
The “trigger” function knows how many jobs need to run, and it can know how long they usually take. Can you make it calculate what range of delays (above I suggested 10 to 100 seconds to start) it should use in order to achieve certain performance goals? These goals could include a specific maximum concurrency, say, or time until the last “process” function finishes.

Can you have it check CloudWatch to see how long the “process” function has actually been taking lately? Is there anything the “process” function could include in its logs that would help the “trigger” function calculate this, such as size of documents?

Queueing theory is the mathematics about queues and “waiting in line”, and it has applicability in many

areas of cloud resource management. You might want to follow up and read about it!

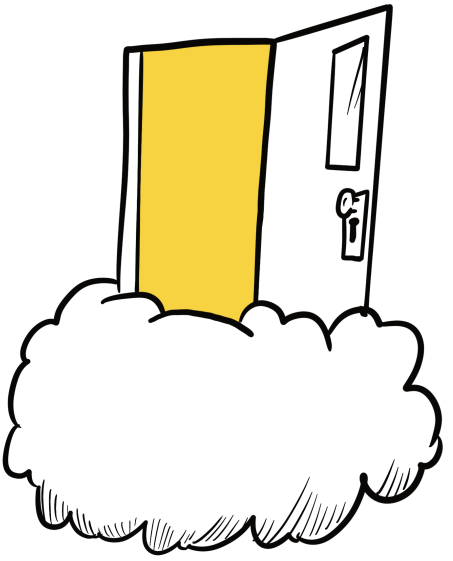
Final takeaways

This challenge shows some real-world tradeoffs between synchronous revenue-producing production code, which is clearly important, and asynchronous delay-tolerant processes that are … also important.

Making the effort to understand actual requirements often clarifies opportunities for simple and effective solutions. I've asked you to do the math for your performance expectations, and then verify the result, and set up ongoing monitoring. We've also demonstrated a few architectural principles -- pub/sub vs queue, jitter, queue behavior -- but there are many more! For a deep dive into how Anmazon applies these principles at scale, check out this article from the Amazon Builder's Library.

Be ready to answer questions! There are many ways to solve these problems, why not do it this other way? Educate yourself about those other ways! This is not a complex or fancy solution, it’s a simple solution using rock-solid cloud services that I’ll never have to fuss with.

Part 3: From challenge to job



You did it. You really did it. Congratulations. You completed the entire Cloud Resume Challenge. You wrote the blog. Maybe you even did a couple of the extra challenges in this book.

And maybe you did all that work just for fun. (I've met a few who have! No judgment, but you're cut from different cloth than most of us, that's for sure.)

No, it's much more likely that you completed the challenge with bigger things in mind. A promotion. A better job. Maybe even your *first* job in cloud. And that achievement is absolutely within your grasp.

While the challenges in this book can be a great advantage that set you apart from other job candidates, you have to know how to use them, and how to fill in the other gaps hiring managers may be interested in.

Let's start off with some general advice on positioning a portfolio site in an interview from my good friend, The Duckbill Group's Chief Cloud Economist Corey Quinn, and then I'll help you put it in context based on your specific career stage.

How to effectively interview for work with a portfolio site

by Corey Quinn

If you've taken the Cloud Resume Challenge, it's reasonable to assume that you're looking to start a career in the world of cloud computing. Congratulations or condolences, to your preference.

There’s a good chance that in hopes of snaring your next great job, you’ve built a portfolio website on top of AWS to show off your cloud computing prowess. Unfortunately, the interview process isn’t as easy as saying, "Here's the site I built, where's my desk?"

Instead, let’s explore how to talk about the site you've built in ways that answer the questions your interviewer has, but likely isn't going to do a great job of articulating.

The three questions interviewers ultimately want you to answer

Every job interview you ever have distills down to three questions, asked in a variety of ways. They come down to your ability, drive, and compatibility.

Cast the questions your interviewer asks through the lens of these high-level questions, and you'll understand where the employer is coming from a lot better.

1. Can you do the job?
Are you crap at computers?

Do you have the raw technical skills needed to do the job? Do you have experience working at similar points of scale? Are you current with the technologies we work with / can become so in a reasonable timeframe?

The hiring manager absolutely needs to have confidence that you can get the job done. It doesn't matter how much they like you, how convenient it would be to hire someone immediately, or how impressive your resume is if you're fundamentally incapable of doing the job that they need done.

2. Will you do the job?
Are you going to rage quit in a month?

Is the role aligned with your interests, or are you likely to job-hop six weeks in and leave the employer back at square one? Will you be happy in the role, or will you hate large parts of it and constantly try to avoid them? Does the job align with your future plans?

Hiring people is expensive. Once they start, it's even more expensive because now other staff needs to devote time and energy to bringing the new hire up to speed. If you're going to quit before the company has the chance to realize that investment, you're not a solution, you're an expensive problem.

3. Can we stand working with you?
Are you a jerk?

Are you going to pick fights with colleagues? Are you going to interrupt people when they're explaining things? Are you going to presume that your coworkers are fools?

If you're amazing at the role but can't collaborate with other people for more than four hours before they threaten to quit, you're not a hire that any reasonable company is willing to make.

How to show off your portfolio site during interviews

The Cloud Resume Challenge was posted in 2020 and continues to be a popular structure for folks new to cloud to demonstrate their skills. The challenge outlines requirements for cloud career hopefuls to go through the process of building a blog on top of AWS and creating a blog post. This includes:

- Passing the entry-level Cloud Practitioner certification.
- Creating an HTML version of your résumé.
- Styling with CSS.
- Hosting on S3.
- Configuring DNS correctly to get TLS working
- Enabling a counter with Javascript that stores its data inside of a database.
- Building an API in Python for the previous two things to communicate.
- Testing for the aforementioned Python.
- Making sure all of this lives in GitHub, via "Infrastructure as Code."
- Putting in place CI/CD for both the frontend stuff as well as the backend.
- Above all else, writing the blog post!

If it wasn't clear, this is a lot of stuff.

It touches basically everything under the sun in terms of cloud skills. It requires research and asking for help. In the process, you'll discover some aspects that naturally align with how you think while other stuff remains a mystery to you.

In the context of an interview, if you say building this site was all easy, absolutely nobody will believe you. Be prepared to talk about what parts were easy for you, which parts were challenging, and

why you made the choices that you did. Be prepared to pull up different sections of the code you wrote (this is part of the reason to keep it in GitHub—easy demonstrations mid-interview!) and show the parts you're particularly proud of. Be prepared to explain any aspect of your code in terms of why it works the way that it does.

Let me underscore the importance of being prepared. It may take you a minute or two to come back up to speed if it's been a few weeks or months since you built your portfolio. Most of us see code we wrote six months ago as having been written by a stranger.

Talk through your thought process during the interview. You want to show the interviewer how you think.

Reminder: Interviews are two-way streets

If your interviewer makes comments along the lines of "that doesn't seem hard to me" or is otherwise dismissive of your portfolio, it's a red flag.

Someone who's universally good at everything contained within the Cloud Resume Challenge is an incredible rarity. Prospective employers who don't recognize that are likely to have incredibly unreasonable expectations for their staff. I'm pretty

good with the cloud stuff myself, but the JavaScript, CSS, and testing portions are all things I'd struggle with if I personally did the challenge.

I absolutely understand being in the position of "I need a job, any job." I spent a lot of time there myself in my early career days! But if you have the luxury of being picky, the first thing I'd veto is obnoxious managers and colleagues—especially those who underestimate the work and workload. The job and its managers should be as good a fit for you as you are for them.

Remember those three questions an interviewer is trying to figure out the answers to? You need to find your own answers to them, too.

1. Can I do the job?
2. Will I do the job?
3. Can I stand working with you?

If the answer to any of these is “no,” you should think long and hard before accepting the job.

Tech takes a backseat to company culture and learning

A lot of folks are very picky about what technologies they'll work with. I get it! If I were taking a cloud job, I'd be far less useful in a shop that used Azure than I would one that used AWS.

But I'll put even that preference in the backseat compared to how I'd consider working somewhere that didn't treat people like human beings.

Technologies come, and technologies go. The danger in tying your identity so closely to a particular stack is that when the technology starts to be supplanted in the market, you're more likely to dig in your heels and resist the change.

It happens to all of us. I started my technical career as a large-scale email systems administrator. The SaaS wave started supplanting email because most companies don't need a dedicated on-site email admin when they're using Gmail or Microsoft 365 to serve their email needs. It was apparent that this wasn't going to be something I could coast on for the next four decades, so it was time to improve my core skillset and embrace something else. I focused on configuration management for a time, using tools like Puppet and SaltStack—and then it happened again. Containerization and "immutable infrastructure," particularly infrastructure as code, changed that particular landscape.

These days I focus on cloud, specifically AWS. If AWS starts losing market share and mind share, it'll be time for me to once again shift to embrace whatever the market is doing.

One thing about technology that stands in stark contrast to many other professions such as accounting, law, civil engineering, and many more: Today's "state of the art" is tomorrow's "how legacy companies do things." You'll be forced to learn new things and experiment wildly as you evolve your career, and so will the companies you work for. That's going to be something you need to take to heart, lest you wind up with 10 years of experience that's really just one year that you repeated 10 times. "Evolve or die, dinosaur," is the way this industry works.

Getting hired without tech experience

For hiring managers, real-world experience trumps everything. Certification, formal education, side projects -- they all have value, but someone who's Done It Before and gotten paid for it will always have the upper hand in an interview.

Don't be discouraged, though. You can absolutely compete and win junior-level cloud jobs without previous IT experience. The Cloud Resume Challenge is your ace in the hole here. Remember, what you're building is real, and it's complex enough that many of those "experienced" people -

even, in some cases, the hiring managers themselves! -- have never done anything like it.

You will, however, likely need a bit broader and deeper background to really pass a cloud interview.

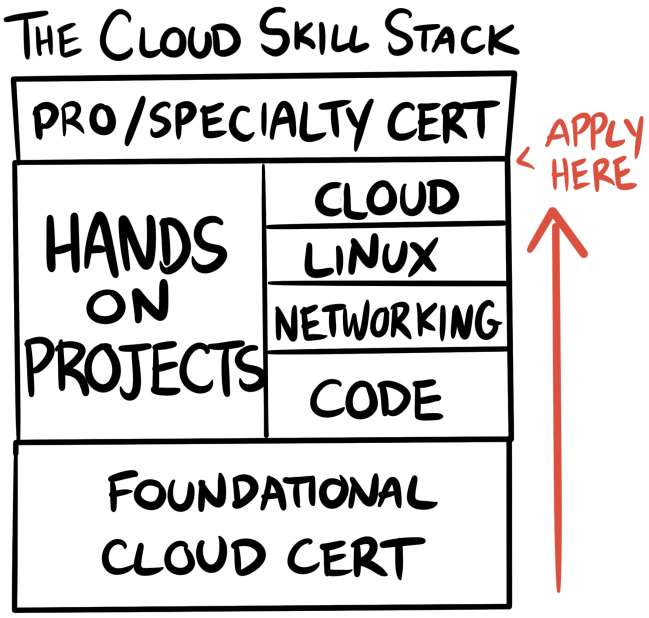
How long will it take me to get hired after completing the Cloud Resume Challenge?

One to six months is the average from career-changers I've heard from; however, this is a biased metric that probably self-selects for success. The odds of getting hired will greatly increase if you work on all elements of the "skill stack" shown in this section.

Check out the detailed post by this Redditor who breaks down all the certs, projects, and skills they worked on over a six-month period before getting hired into a $70k/year job. One interesting feature you may note in that timeline: they didn't attempt the Cloud Resume Challenge until fairly late in the six-month window, getting hired about 1.5 months later. In the comment section, this person confirmed that they held off on the challenge for so long because they just didn't feel ready for it yet -- and this is totally valid.

Building your skill stack

I encourage aspiring cloud engineers to build a "skill stack" that looks like the following:



Cloud is the latest in a long line of IT paradigms. But though it's highly in demand, "cloud" isn't exactly a skill on its own. Hiring managers are going to want to see that you have knowledge in several foundational areas.

Code

You must be able to write basic scripts. I recommend Python as a programming language to learn — it's used on cloud teams the world over. You don't need to be a rock star at algorithms and data structures, but you should know how to manipulate data objects and interact with cloud services.

Networking
You will be expected to have solid familiarity with how the internet works. DNS, TCP/IP, and certificates — these protocols will rule your life in the cloud, where everything is made up of services talking to each other over a network.

Linux
Linux is just as relevant in the cloud as it was in old-school sysadmin jobs. Know how to navigate the Linux filesystem, use Vim and terminal commands, and a bit about containerization.

When to apply

Once you have built a hands-on project that touches each of the basics listed above, I would say go ahead and start applying for associate-level cloud support and engineering roles. You should continue to improve your skills and level up your resume during this process by studying for a professional- or specialty-level certification on your chosen cloud, like the GCP Professional Architect.

It's okay if you haven't taken the exam yet; hiring managers like to see that you are self-motivated and keep learning on your own.

It may feel uncomfortable to start interviewing at this stage, and you may encounter some early rejection. But the weaknesses these interviews expose will help you keep a short feedback loop to go back and polish up those key coding, Linux, and networking skills.

Plus, now that you've been through the foundational study process, you'll have a better sense of where you might want to specialize, and why. You can share that information in an interview – along with the war stories you picked up from your portfolio projects.

Will the transition to your first cloud job take a lot of work? Quite possibly. You can expect to spend some late nights at the kitchen table and some time down frustrating rabbit holes on StackOverflow. But a few months of dedication here is going to pay lucrative dividends down the road.

Because if you study smart and build your skill stack the way I'm showing you, you won't have to spam out thousands of resumes and wonder why nobody's calling you back, no matter how many certifications you've lined up. The right combination of certs and hands-on builder experience will stand

out among the crowd, and that's the fastest way to get ahead in the cloud.

Jerry's story: From respiratory therapist to Associate Solutions Architect at AWS

How long it took Jerry to compete the Cloud Resume Challenge: *About 20 hours over the course of 1 week*

How long it took Jerry to get a cloud job after completing the challenge: *1 month*

So you had never worked in tech before tackling the challenge?

I worked nearly 14 years as a respiratory therapist and in a healthcare management role.

I had certifications and other small projects for my portfolio but I wanted to accomplish a build that would give me hands-on experience with a broad range of services. Something I could explain in-depth and walkthrough in an interview situation.

How much did the certs help you get hired, as opposed to the project experience?

I had 4 AWS certifications. I would say it was a bonus to have getting my name in front of the

right people but ultimately getting hands-on and understanding the HOW and WHY was the most beneficial aspect. For anyone who asks for advice that's the first thing I mention. Do the Cloud Resume Challenge or compatible challenge and get your hands on the console and CLI.

How did you use the challenge during interviews?

When interviewing with Amazon, the Cloud Resume Challenge was a big talking point for me. When going through the interview "loop", everyone seemed to be impressed with the range of services used within this one project.

I also had to build something myself to demo during the interview process and included some skills I got from the Challenge to build a serverless transcoding pipeline integrated with my Synology NAS, Elastic Transcoder and Lambda functions.

What do you wish someone had told you during that 4-month job search?

Don't. Give. UP! Keep pushing. No matter how hard someone says it might be, no matter how frustrated you may get at times, keep learning

and stay positive. Imposter syndrome is a real thing and we ALL deal with it at some point in our careers.

Get hands-on experience building and architecting, anticipate failure and re-build again until you nail that project/build. As AWS CTO Werner Vogels famously said, “everything fails all the time." Keep pushing, keep building, and keep your head up in the clouds.

Getting hired with IT experience, but no cloud experience

Often, when I give presentations about the Cloud Resume Challenge, I tend to focus on the stories of people who used it to find their way into cloud jobs from completely different careers, like plumbing or HR. That's because these stories are surprising!

But inevitably, after those talks people tell me: "I want to get into cloud too, but I have 15 years of experience in IT. Can the challenge work for me?"

Absolutely, it can.

Remember, almost 40% of people who undertake the challenge have more than 3 years of professional IT experience. So it can still be a valuable addition to your resume even if you're not brand new to the industry.

Stacy's story: From IT Service Desk Manager to Cloud Operations Engineer
How long it took Stacy to complete the Cloud Resume Challenge**:** *50 or so hours, spread over 6 weeks*

How long it took Stacy to get a cloud job after completing the challenge: *1 month*

What was your background in IT before turning to cloud?

I'd been working in IT for several years with a Windows administration focus, but I wanted to specialize in cloud for career growth.

When I found the Cloud Resume Challenge, I thought it looked like a great way to learn cloud tech by getting hands-on experience and completing a project that could be added to a resume or used as a talking point in interviews.

How did you use the challenge during interviews?

When I was asked about AWS experience, what AWS services I was familiar with, or technical questions (the difference between public vs. private subnets, describe a 3-tier architecture), I was able to bring up my work in the challenge as examples. Due to the wide variety of commonly used AWS services that were incorporated into the challenge, it was easy to use that as a talking point.

Any advice for other upskillers such as yourself?

Don't give up! The jobs are out there, but you need to find the right environment for growth. A lot of cloud jobs are geared towards senior-level people with years of experience, but there are companies out there that are willing to grow their own.

Doing the Cloud Resume Challenge will set you apart and show that you have the curiosity, resilience, and motivation to make a career in cloud.

Also, while you are interviewing, start learning Terraform and Linux administration. Those two skills will help you immensely both during the interview process and on the job.

That said, you'll have different challenges - and different advantages - landing a cloud engineering role as a mid-career IT admin or software developer compared to someone who's just starting out.

Exploiting your "unfair advantage"

In the startup world, venture capitalists like to talk about finding your "unfair advantage". What is the thing you do better than anybody else, the thing nobody can compete with?

If you already work in some form of tech, your unfair advantage in finding a cloud job may simply be that *you already work in tech.* You have more connections and more situational awareness. IT is in your blood.

Most importantly, you may have opportunities to get hands-on with cloud at your current work. Is your team doing a migration project that requires somebody to know a little bit about EC2 or VPC? Are you an integration engineer with opportunities to run some scripts in the cloud? Push yourself, get outside your comfort zone, and build your resume.

But *be realistic about this*. If the cloud experience you're looking for can't be had in your current role, don't hang around forever waiting for it. It may make sense to start looking for a new position that may not have the fancy "Cloud Architect" title, but will let you get hands-on with AWS or Azure in some capacity. The type of role where previous cloud experience is a "nice-to-have", not a "must-have".

The Cloud Resume Challenge may be a nice assist in helping you stand out for that role - it could be your "unfair advantage" at a crucial moment in your career.

Avoiding a temporary step back

One challenge of revamping your skill set for cloud: you may find that many of the opportunities you get as a junior cloud engineer are a financial step backwards from what you were making in a less future-proof, but more senior role.

Depending on what you're doing right now, it may be difficult to find a cloud-related role that pays what you need. It's a hard sell to go from senior sysadmin to junior cloud engineer. That's why I'm recommending that you try to build cloud skills on the job, rather than starting from scratch.

It's also true that, due to the explosion of remote work combined with the huge demand for cloud talent, more and more companies are opening up to the idea of hiring consultants for cloud project work. Startups are especially noted for doing this, usually with low friction and a high degree of autonomy.

Keep an eye out for side work that you can do to build marketable experience without taking a

lower-paying day job. (I curate many of these opportunities in my "Best Jobs in Cloud" newsletter, which you should be receiving automatically now that you've purchased this book.)

Networking your way to a job

I have hired many people over my 10+ years in technology, and helped others transition into their first tech jobs through the 16-month life of the Cloud Resume Challenge. In all that time, the worst way to find a cloud job that I have seen is by playing a game I call Resume Roulette.

Resume Roulette: a losing game

Resume Roulette is easy to play but very hard to win. Here are the rules:

1. Wait for random companies to post public job listings
2. Spam out a bunch of applications
3. Hope that some stranger likes your resume enough to set up an interview
4. If you don’t strike it lucky, repeat steps 1-3 indefinitely until you get hired or burn out

Do people get hired by doing this? Sure, every day. You can find inspiring stories on LinkedIn of people who sent out 190 job applications, got 10 interviews

and 1 job offer. But far more get discouraged and give up long before then. I know I would.

In fact, when I do hear of someone beating the odds and getting hired after sending out hundreds of cold resumes, I don't think "Wow, that's great perseverance" (although it certainly is) — I think "We are celebrating a broken process."

How hiring happens

Let me tell you what happens inside most companies when they list a public job posting:

1. 150 random people submit their resumes, some of which are pretty good and some of which are pretty bad. Hard to say without really digging in.
2. Three current employees know somebody who would be great in the role, and they submit referrals with a glowing recommendation: "I worked with her at Company X! She would kill at this!"

If you're a recruiter (or especially a time-strapped hiring manager), which resumes are you going to move to the top of the pile?

Many companies offer referral bonuses to their employees for exactly this reason: the best hire is often someone your existing team already knows.

And it's much more efficient to vet those people than to do a phone screen with 150 total unknowns.

(Yes, there are a few companies out there, the Googles of the world, that have such a massive recruitment engine that they can reliably crank through hundreds of thousands of unsolicited resumes every year. But you're fighting a numbers game there as much as anywhere else – you're just more likely to get a formal rejection, rather than having your application sit ignored for six months.)

Network Bets: a better game

But that doesn't mean you're stuck on the outside looking in. You should not have to send out 190 job applications to get a cloud job. There is MASSIVE demand for qualified professionals in this field! If you know your stuff, companies should be basically coming to you.

But to come to you, they have to know about you!

So in my experience, the BEST way to get a cloud engineering job is to play a different game, one we might call Network Bets, a game with many ways to win but just two rules:

1. Own your credibility
2. Build connections

Owning your credibility

I covered foundational cloud skills in more detail above, so I won't reiterate everything here, but if you're brand-new to cloud you need to make sure you have:

- A solid ability to write code in Python or another common back-end language
- Hands-on familiarity with Linux and network fundamentals
- A foundational cloud certification or two

If you don't get your feet under you here, you're just going to be demoralized coming out of interviews. I don't want you to be demoralized! I want you to skip to the head of the line.

Getting proficient at code, networks, and Linux (not a rockstar! Just the basics!) is your cheat code. Seriously. Do it. You can get there faster than you think. Projects help.

Building connections

Yes, this is the part where we talk about networking. No, not about getting your CCNA, I'm talking about knowing the right people to work with. I've noticed that a lot of engineers tend to bristle when faced with the idea that they might have to network in order to land a good job: "My skills

should stand on their own. My career shouldn't depend on having connections; that's not fair."

But what they are missing is that connections go both ways. You are more than just an interchangeable YAML-generation unit. You are a person with likes and dislikes and personality quirks of your own. There are tech jobs out there that would be toxic for you, where you would not be happy or productive — and plenty more where the work itself simply isn't interesting or beneficial to your future career.

And when you fling random job applications into companies you know nothing about, you face the very real possibility that even if you do win the lottery and get hired, it'll be on a team that is nothing but bad news.

Networking is your chance to vet the job market at the same time they are vetting you. It lets you figure out the good people to work with, at companies where you will fit in and be set up for success.

But I'm an introvert … I can't network!

I'm an introvert too! In fact, I'd venture to guess the majority of people in technology are. The amazing thing about building public credibility in tech is that you don't have to go around glad-handing, passing

out business cards or whatever. (Not that that would even work!)

As AWS Hero and fellow introvert Alex DeBrie points out, the internet lets you connect with people very effectively through less draining, more asynchronous means.

As long as you are demonstrating passion for your skills in public, you'll find yourself encountering great people and opportunities.

So let's run through three practical ways to make that happen. Here are my cheat codes for networking your way to a cloud job that will actually be good for you and your career:

1. Find a new niche
If you're new to the cloud, you stand an obvious disadvantage against more experienced people. Many people try to close this gap with certifications, but the most common entry-level credentials are not that impressive to hiring managers. (You have an A+ cert? Cool, so do 100 million other people!)

So the smart move is to stop playing the game everyone else is playing — to skip to the head of the line and get good at cloud services that are so new, nobody has that much experience in them. After all, nobody can ask for 5 years of experience in a technology that's only existed for 18 months.

I'm not talking about becoming an expert in a huge, broad area like the entire Kubernetes ecosystem. You can find a niche here that is really specific. Here is a list of some hot cloud tools and services that are in desperate need of more community attention. There are many, many others; this is just what I could write down in 60 seconds.

1. AWS AppSync/Amplify (managed GraphQL gateway and associated front-end framework)
2. The AWS Code* stack (CodeBuild, CodeDeploy, CodePipeline)
3. Serverless workflow services (AWS Step Functions, Azure Logic Apps)
4. A managed Kubernetes service like GKE, AKS or EKS
5. Popular Infrastructure as Code tools like Terraform or the AWS CDK

If you get good at any one of these 5 things, you are guaranteed to make some interesting friends and raise your profile as a cloud technologist, as long as you also …

2. Plug into the community

As you start getting hands-on with your chosen technology, write about what you are learning. I suggest using an established developer blogging platform with some built-in network effects, like

Hashnode or dev.to. For example, if the service you are learning about releases a new feature, write up a short exploration of what it does and why you are excited about it.

Then, SHARE those good vibes! And tag the creators of the feature you are excited about. Devs love to see their good work recognized, and they will welcome you with open arms.

You will also start to discover Slack and Discord channels, Twitter hashtags, etc, that your chosen community rallies around. Engage with these outlets and note who the influencers are — the people you would want to be your mentors if you knew them.

I highly recommend joining Twitter and following key influencers in your niche. Engage with them, reply to their tweets with questions. People remember people who engage with them positively and helpfully. And you'll start to branch out into other connections as well.

Then, when you see a job opportunity pop up inside that community (which is guaranteed to happen)…

3. Ask for referrals
This feels really uncomfortable at first, but it's way easier than it seems. You can literally just message people who work at the company you are interested

in and politely ask for a referral for the open job position. Without pestering, you could even ask if they have any openings that are not publicly listed. The worst they can do is say no.

But remember – they're likely incentivized by their company to say yes. And they're much more likely to give you that boost if you've interacted with them by building and sharing work in their space in the past. If you've blogged or built open-source projects, they can pass that along to their internal recruiters. And familiarity with them gives you a shared basis of trust to know if this role and company would really be a good fit for you.

Again, my "Best Jobs in Cloud" newsletter, which you should be automatically subscribed to if you bought this book, makes referrals even easier - I curate referral contacts directly from hiring managers and all you have to do is hit reply!

I'd also be remiss not to recommend The Coding Career Handbook, written by ex-AWS and Netlify developer (and friend of the challenge) Shawn "swyx" Wang. His book goes into much more detail about how to play and win the online networking game.

Play the long game – it's a shortcut

Landing a job through the Network Bets process — nailing down the fundamentals, finding a cloud niche, and getting plugged into the community — is not necessarily fast. It could take six months to a year or more. But that's a lot faster than getting a college degree. Plus, you can do it very cheaply and in your spare time – while building connections that ultimately will pay off for decades to come.

On the contrary, Resume Roulette has no barrier to entry, but is a bad game because it has so many poor outcomes:

1. You compete with everyone else on the job market, decreasing your chances of getting hired
2. You go months or years without getting hired, and you get so disillusioned that you stop searching
3. You actually *do* get hired, but in a job that's not a good fit for you

So be active, not passive. Own your credibility. And create a niche for yourself, don't wait for the perfect opportunity to open up. That's the best way to find a cloud job that becomes the foundation for an awesome career.

Afterword: Paying It Forward

The door to every big career opportunity has two sides. The outside says "Hard work", or maybe just "Luck."

The inside says "Generosity."

Every one of us working in cloud today, senior or junior, got started in our career because someone took a chance. Someone believed we could succeed. And then we busted our tails to prove them right.

I started the Cloud Resume Challenge because I wanted to pay forward that generosity to others. People that didn't look like the stereotypical software engineer.

A commercial plumber in downtown Atlanta. A help desk administrator on a Pacific island. A front-line healthcare worker. A single mom returning to the workforce. All of them brilliant, self-motivated, fast-learning people. Every one of them just needed that little nudge, a quick guide, the right connection, to set them on their way.

You see, the Cloud Resume Challenge is about more than just you and me, a writer and a reader. It's now a community initiative that has touched thousands of people. It's a movement.

It's about the hiring manager who realizes they need to build an apprenticeship program to onboard more junior engineers. The senior engineer who mentors a challenger. The kid who decides to leave a dead-end degree program to build cloud skills.

And every blog post you write, every project you share, spreads the movement a little further.

When you support the Cloud Resume Challenge, you help to teach the world that a great cloud engineer can come from anywhere, from any background. In a small way, you are helping to build the future of this industry.

If the Cloud Resume Challenge helps you enter a new phase of your career, don't forget to hold the door open for those behind you. The challenge has grown, but the spirit remains: senior, junior, career-changer or upskiller, we are all in this together.

Forrest Brazeal
August, 2021

Acknowledgements

My thanks to A Cloud Guru, who not only supported the underlying mission of the challenge by creating the spin-off "Cloud Guru Challenges", but also graciously permitted me to incorporate material from two of my ACG blog posts in this text. Keep being awesome, cloud gurus!

Daniel Dersch, Corey Quinn, Jennine Townsend (times 2!), and Justin Wheeler contributed essays or challenges to the book. I owe you all a drink the next time that's a thing people can do.

The entire Cloud Resume Challenge community, now more than 1500 strong, assisted in the creation of this book by responding to surveys, offering feedback on the challenge, and creating resources that everyone can use. Special thanks to Nana, Stacy, Daniel, Jerry, and Jakob for their featured testimonials. I'm also especially grateful for the tireless community contributions of Lou Bichard.