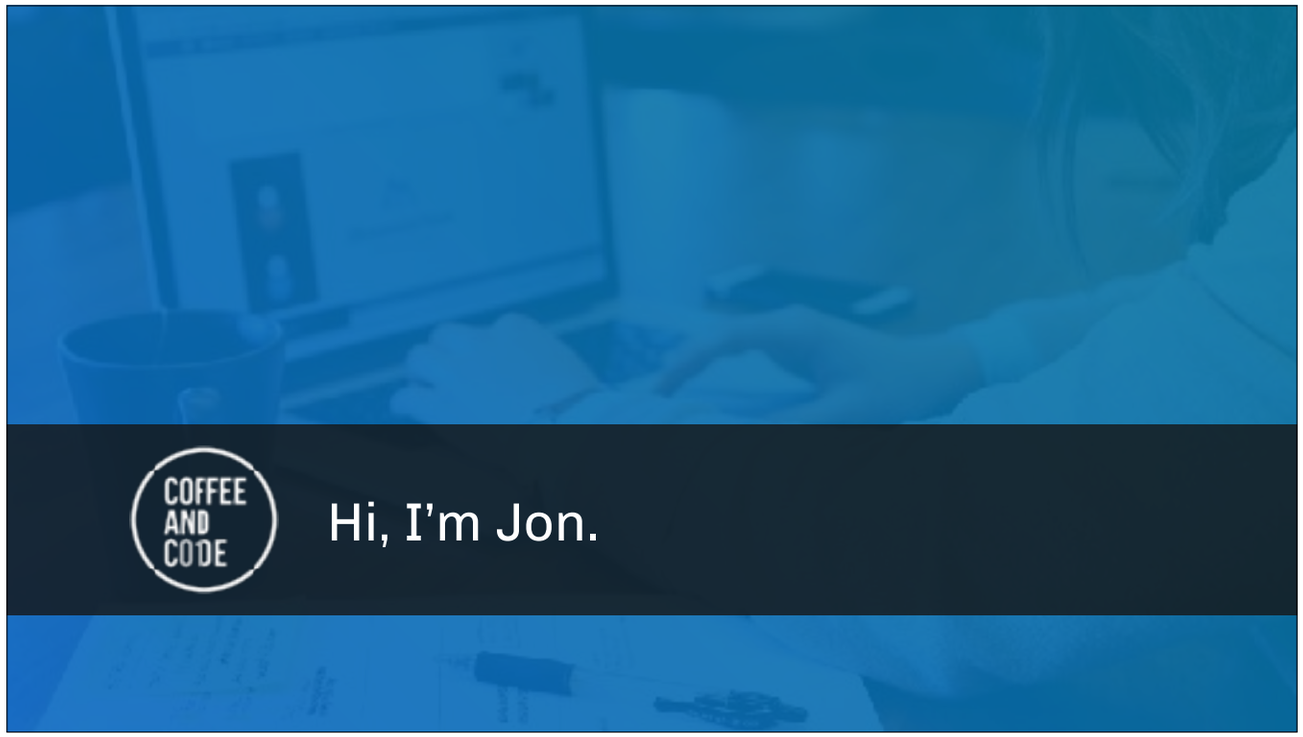
FYI, this talk is not a walkthrough for deploying code to AWS. This is more of a story about a client project we had, what we used to solve the technical problems, and some things to think about so you deliver better projects. If you’re interested in getting your hands dirty, you can check out the “Scale Your Node Application, Skip the Infrastructure” talk that’s going on now in the Guava + Tamarind rooms.

Overview (don’t say out loud):

I was asked to build a fuzzytext search interface for information stored in the SEC’s massive Edgar database which holds all of the electronic documents filed with the SEC. By leveraging managed ElasticSearch, S3 for document storage, and the asynchronous job processing power of AWS Lambda I was able to build a solution that required absolutely no ongoing server maintenance for my client. In this talk I’ll explain how I was able to:

– parse gigabytes of info without IP activity restrictions
– provide an easy way to scale or disable the application
– continuously monitor parsing activity and application health

You will learn about the different services I utilized with their strengths and weaknesses as well as alternative services like Iron.io which allows you to write code in many different languages. I’ll also talk about different ways async processing can be applied to other situations such as managing contact forms for static websites as FormKeep does.



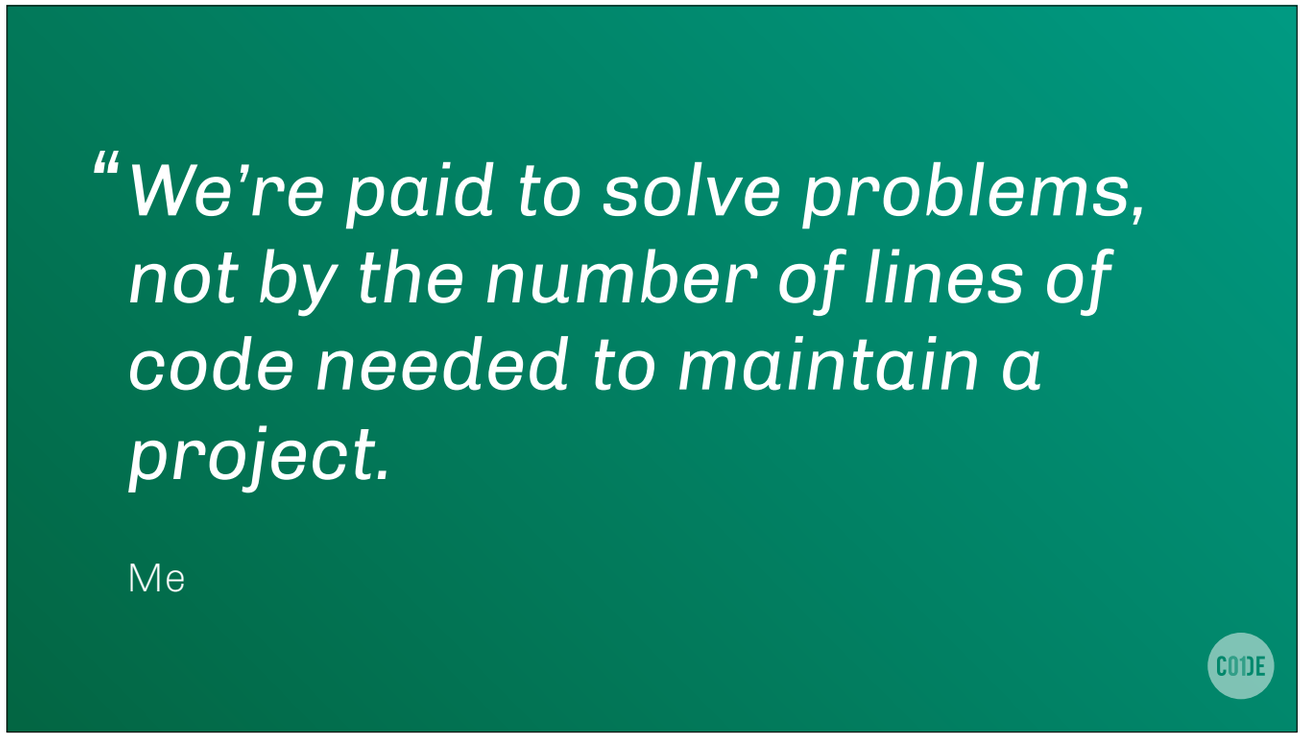
Hi, I’m Jon, owner of Coffee and Code, software development agency in Akron, OH. We’ve been writing code since 2005, delivering custom software solutions over the past decade over products and services on behalf of our clients across many different business sizes and industries.

I’m also anxiously awaiting the birth of my first child, so if I answer the phone and run out of the room to make the delivery in time, feel free to send someone up to read the rest of my slides for me.

Today, I’d like to talk to you about how we used AWS Lambda to make a large portion of a client’s project less complicated, easier to maintain, and easier to scale without our client requiring significant technical knowledge to keep it up and running.

So let’s agree on one thing…

Code is complicated. Or better yet, the applications that we write are complicated. Our jobs may seem easy to us at times, but we have the difficult job of determining how ideas become tangible. Our industry affords us with a lot of ways to do that, and learning new things can be challenging and make us feel intelligent.



But at the end of the day, we’re paid by the problems we solve, not by the number of lines of code needed to maintain a project. Sometimes those problems require owning everything in the stack, but often times it won’t.

I personally like to be responsible for as little as possible. *Did I mention I’ll soon be a new father?* It makes projects easier to maintain, easier for other’s to pick up, and provides a better deliverable for the clients we write code for.

That said, I’m always looking for ways I can simplify the development & deployment process so I can focus on writing the smallest amount of code necessary to meet our goals.

What is Lambda?

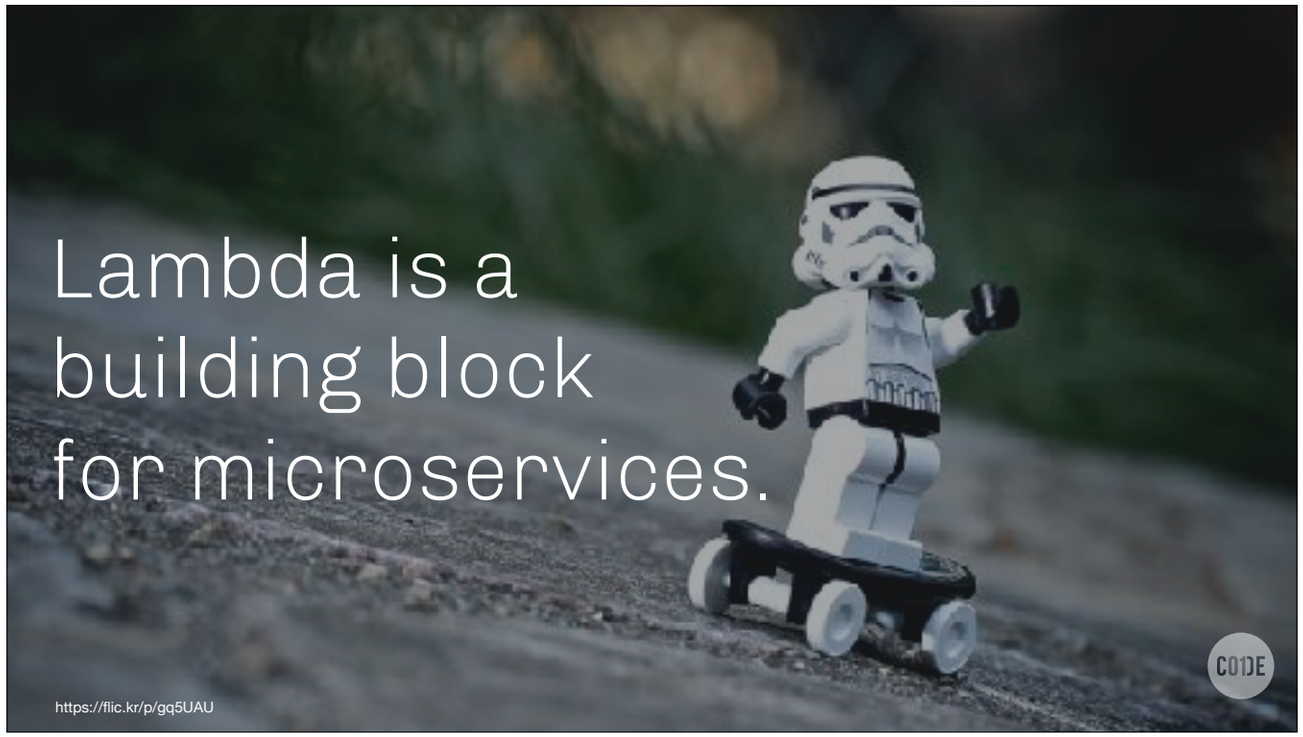
- pay for what you use
- no servers to manage
- runs in response to AWS events
- integrates with AWS API Gateway
- runs Node, Java, Python, and C# natively
- automatic scaling and fault tolerance

CODE

Main focus will be on AWS Lambda.
“events” are triggered by Amazon S3 bucket changes, Amazon DynamoDB table, Amazon Kinesis stream, or Amazon SNS notification.

Can run binaries that are built for the same server. Can try using Docker project for AWS Lambda to build resources.

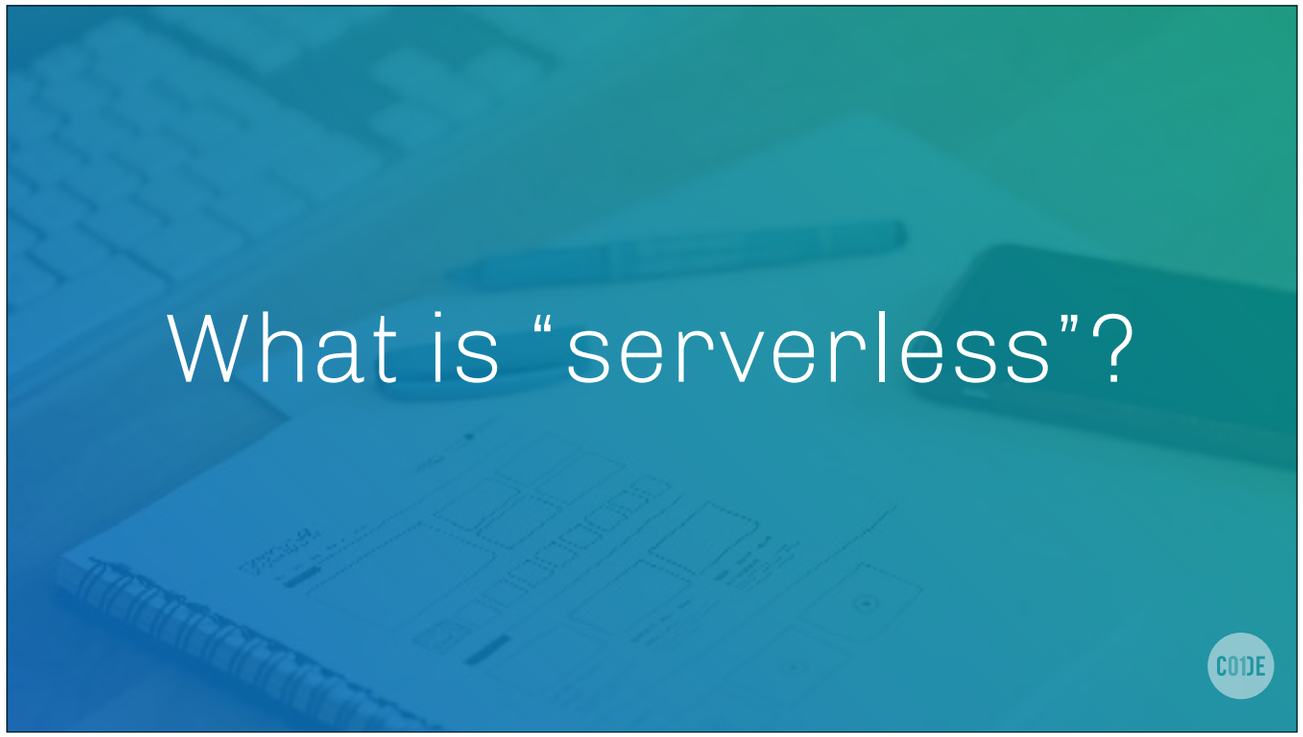
Can be a building block of microservices.



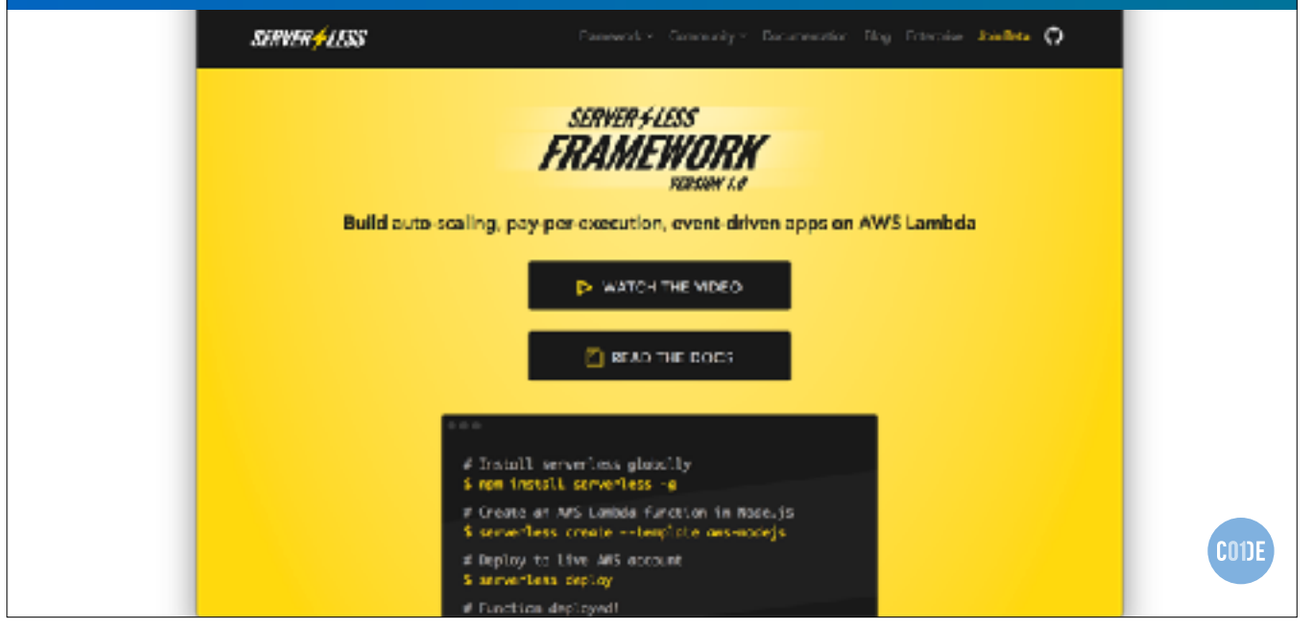
Just like a LEGO model includes multiple parts with different properties, Lambda, API Gateway, Dynamo DB, and the other AWS suite of applications can be used as specialized tools to help build distributed applications.

Lambda is just one of the LEGO blocks we can utilize to help build our applications.

https://www.flickr.com/photos/clement127/10117904824/

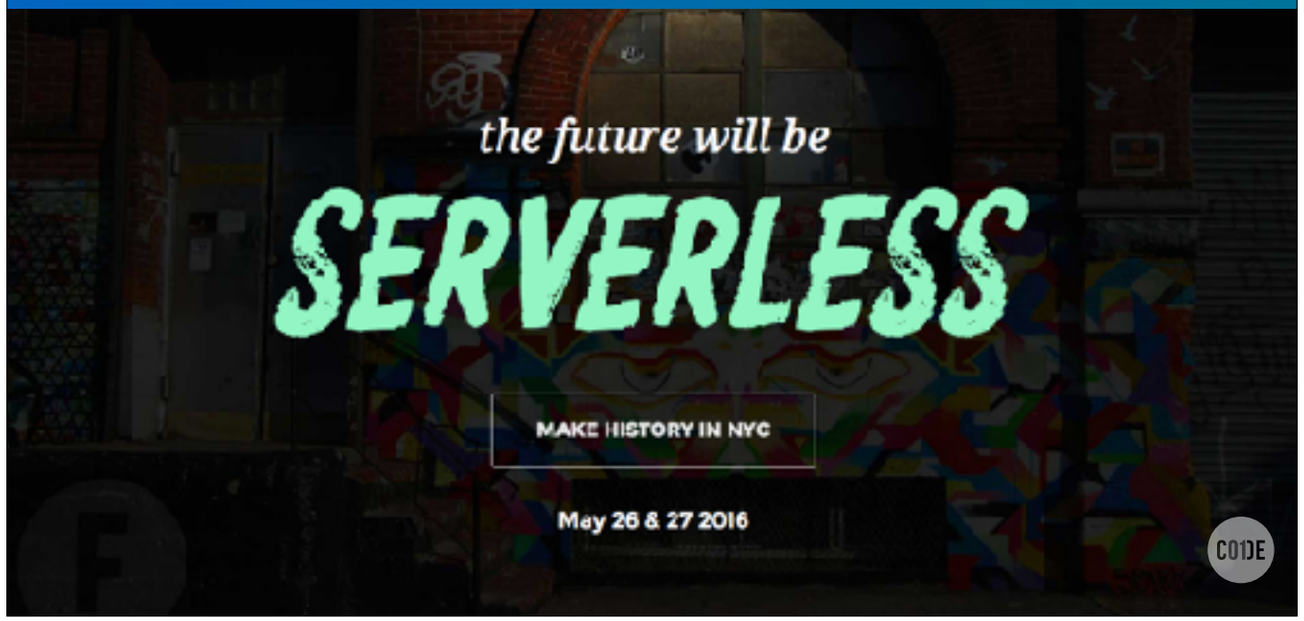

What is “serverless”?
CODE

Serverless is a framework



serverless is a Node application framework for creating Lambda functions and tying them to http endpoints via AWS API Gateway. It has aspirations of connecting to Azure Functions and Google CloudFunctions as well. We'll talk about it later in the tooling section as it seems to be the most popular AWS Lambda code packaging and deployment.

Serverless is a conference

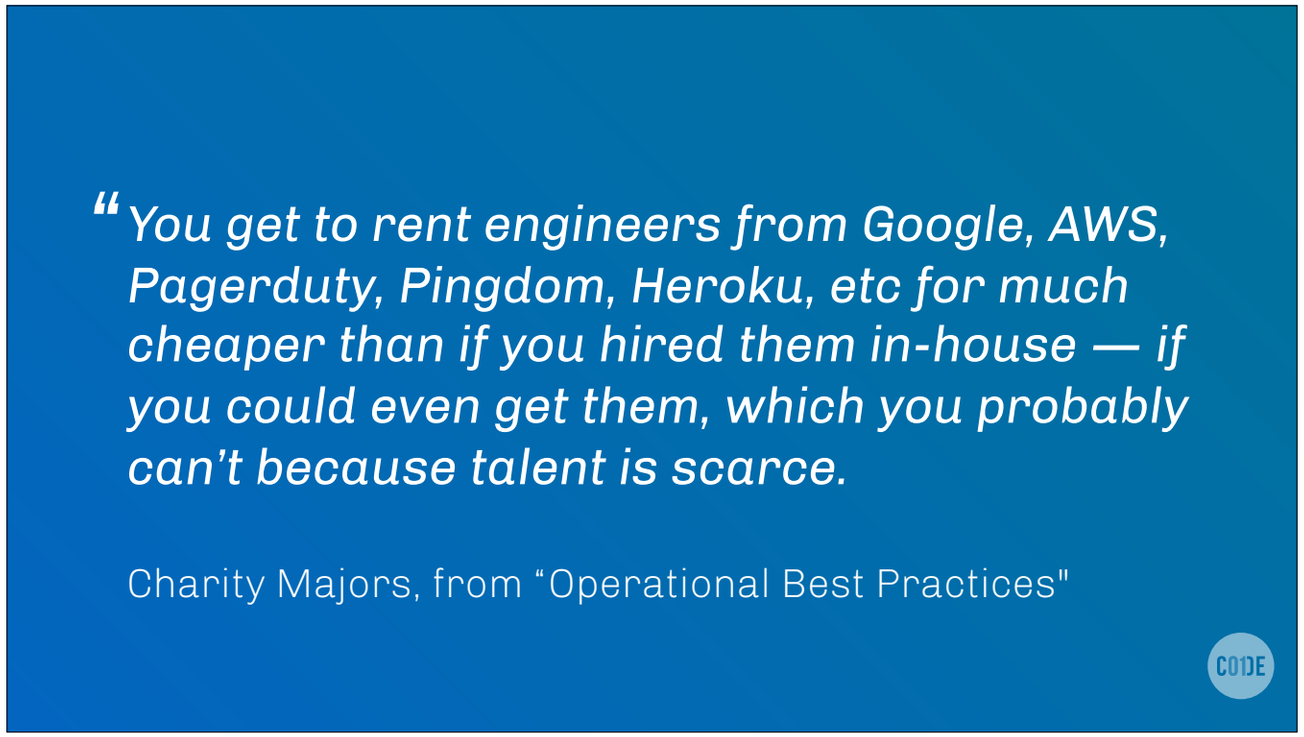


Apparently, this “serverless” is so popular, it has it’s own conference.

Serverless is a buzzword

Unfortunately, it’s mostly a buzzword. Of course there are servers being used to run your code. So if we can look beyond one of the hardest things in computer science which is naming things, we can get to the meat of what serverless is really about.

It’s a mindset, a potentially different way of thinking about how you will construct your next project. At its heart, being “serverless” depends on relying on third party services to leverage the amazing things other people built to make your amazing things easier to build and more robust.



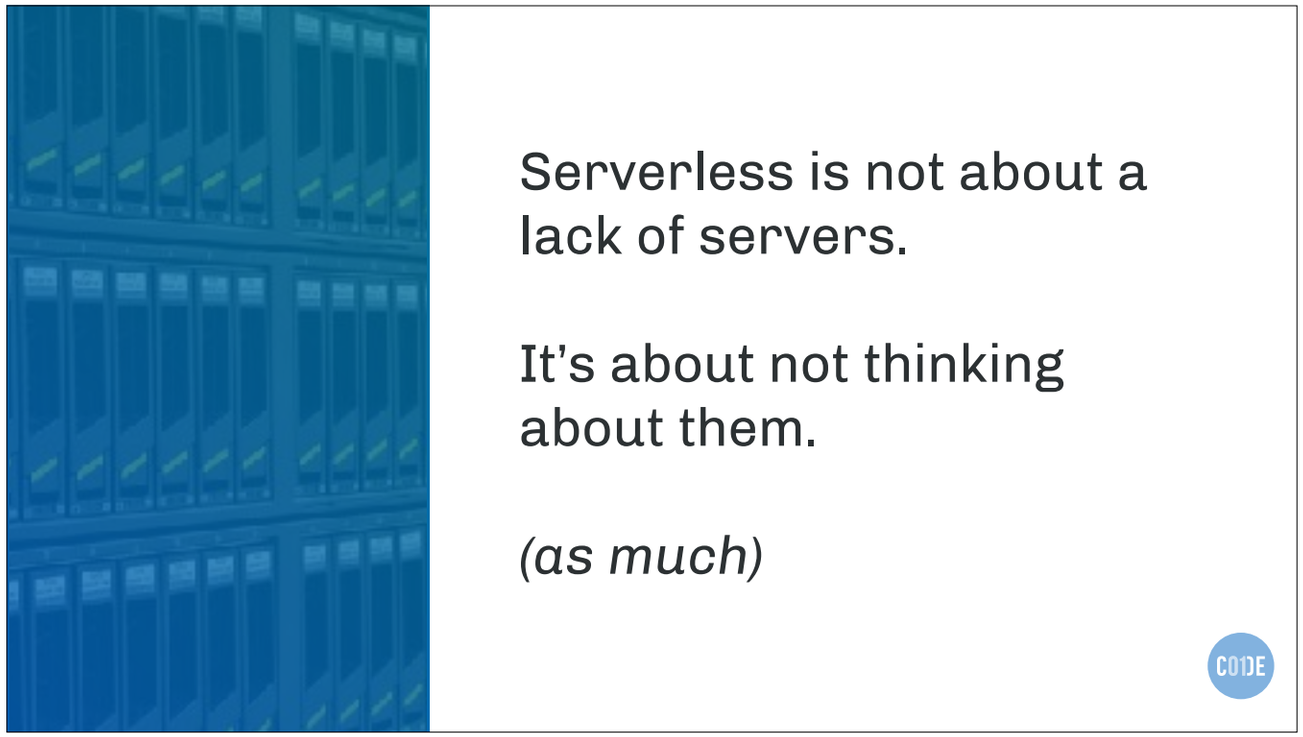
Charity Majors, a CTO/devops engineer/cofounder/speaker/awesome person who has written quite bluntly on operations related buzzwords like “serverless” and “devops”. In her post entitled “Operational Best Practices” she talks about the great powers and responsibilities that come when leveraging software as a service.

(read quote)

She then continues…

> *But the flip side of this is that application engineers need to get better at thinking in traditionally operations-oriented ways about reliability, architecture, instrumentation, visibility, security, and storage.*
>
> Charity Majors, from “Operational Best Practices"

https://charity.wtf/2016/05/31/operational-best-practices-serverless/



So remember, serverless is a mindset.

It’s not about a lack of servers, it’s about not thinking about them (as much). But with these new powers, you’ll face some obstacles you may not be used to in regards to application logs, visibility when things go wrong, and hard limits on resources that can effect how long your code can run.

Now more than a few of you may be thinking that letting another company manage a major piece of your application is a bit of a scary experience. And I can completely empathize with you. It’s true, 3rd party services are not all rainbows and puppies, but if you think about it I’m sure you’ll come up with some other services you depend on quite a bit.

We've been outsourcing physical server infrastructure and email services for years. I for one never want to return to the days when I had to manage an onsite mail server or purchase physical hardware.

Just remember, it is ok to be a little paranoid at times.

> *“Create a custom search interface to the SEC’s Reg A, Form C, and Form D filing information.*
>
> The Client

Objectives

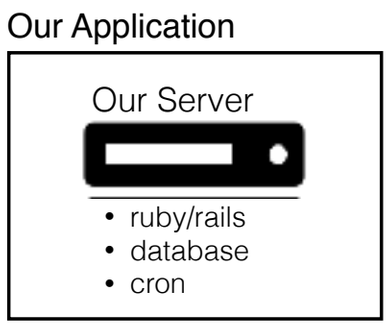
- Parse html & xml pages on Edgar
- Avoid IP traffic restrictions
- Store massaged information for searching
- Archive filing attachments

So how would you build it?

Scaffold a rails project, create some background job processing, call it a day.

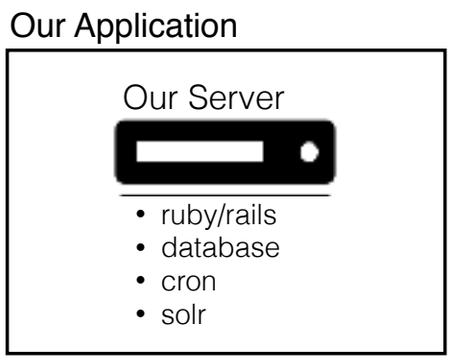
Without some forethought, one might dive right into the server side code base they are comfortable with, but we wanted to think a bit more about what we could deliver and who we were delivering it to.

Example Setup



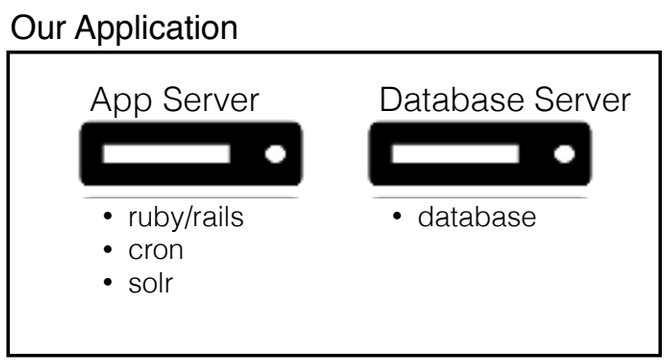
We write ruby code that goes on a server that has our database and a Linux OS installed so we can run our rake tasks on a schedule.

Example Setup



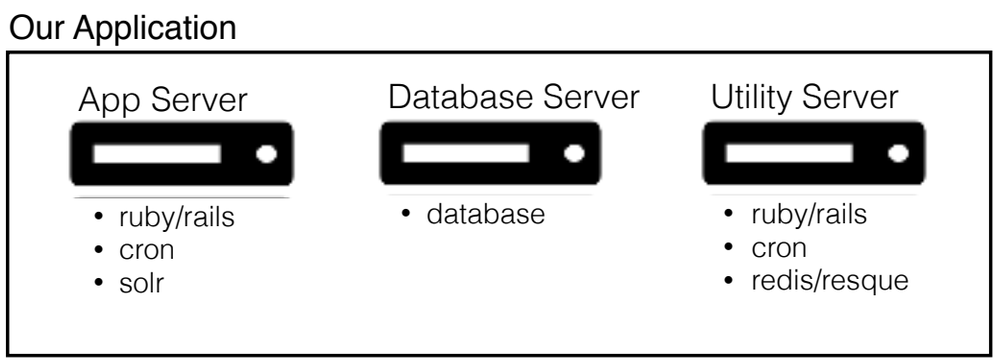
We write ruby code that goes on a server that has our database and a Linux OS installed so we can run our rake tasks on a schedule.

Example Setup



We write ruby code that goes on a server that has our database and a Linux OS installed so we can run our rake tasks on a schedule.

Example Setup

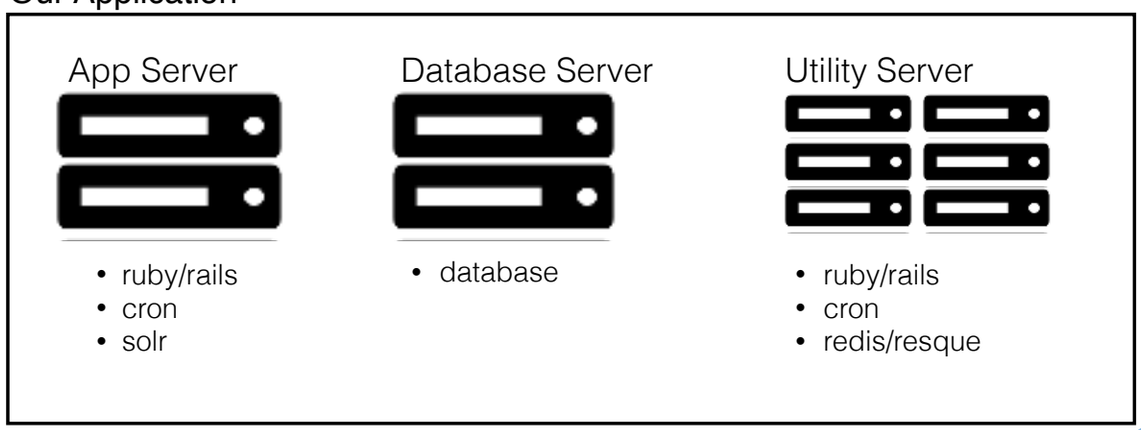


CODE

We write ruby code that goes on a server that has our database and a Linux OS installed so we can run our rake tasks on a schedule.

Example Setup

Our Application



CODE

We write ruby code that goes on a server that has our database and a Linux OS installed so we can run our rake tasks on a schedule.

All of a sudden, you’re back in the land of logos and software overload. Think about all of the code you need to support your application beyond accomplishing what the client originally requested.

“Hidden” Objectives

- Edgar is ~~a horrible piece of software~~ not friendly to work with
- No tech team to hand off project to
- Unknown number of filings to process
- Insight into application errors
- Small, focused codebase
- Easy to maintain

Then we have the our “hidden objectives”, or the items that are never stated by the client but which will ultimately make the project a larger success. We’re a 3 person shop at Coffee and Code and the only reason we’ve existed for over a decade is that we make clients very happy. This is extremely important to us.

Edgar / SEC has severely outdated tech stack. Goes down outside 9-5 M-F, no API, tech staff not very helpful (figure it out yourself mentality). Gigs of information to process.

Architecture

- **Lambda** - Background and async job processing
- **ElasticSearch** - Data storage and search interface
- **S3** - Storage and web hosting of archived form filings
- **SES** - Email notifications and support requests
- **SQS** - Queueing of data processing requests
- **CloudWatch** - Logging and monitoring

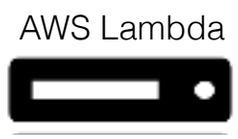
- **Rollbar** - Error logging and alerting
- **DigitalOcean** - Public website hosting

CODE

We values reliability for the client over ownership of the architecture. We wanted to compose existing building blocks to meet the project's needs with a minimal code footprint from us.

We have two parts: the parsing logic and the client facing website.

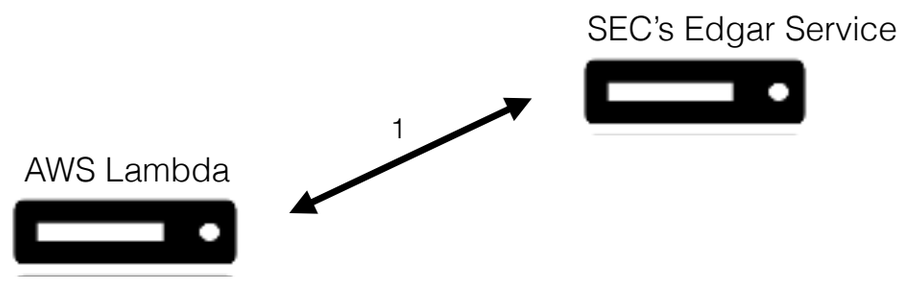
Lambda 1: Preparation



1. Lambda job runs every 5 minutes
2. Request new filings from Edgar
3. Parse results and enqueue identifier for later processing

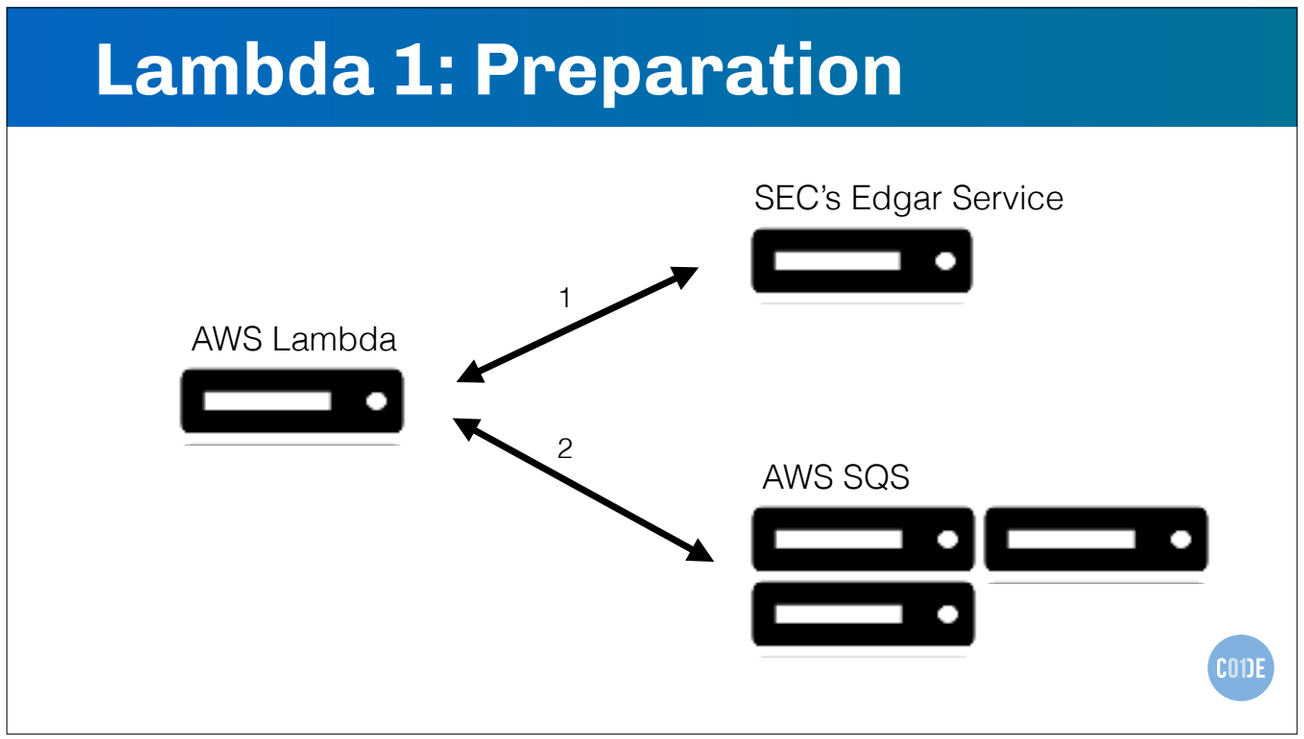
The first Lambda method is kicked off by a CloudWatch scheduled event which is like Cron for AWS. It runs every few minutes and which runs the Lambda function whose job is to look for new filings and queue them up for processing by the second Lambda method we'll discuss next.

Lambda 1: Preparation



1. Lambda job runs every 5 minutes
2. Request new filings from Edgar
3. Parse results and enqueue identifier for later processing

The first Lambda method is kicked off by a CloudWatch scheduled event which is like Cron for AWS. It runs every few minutes and which runs the Lambda function whose job is to look for new filings and queue them up for processing by the second Lambda method we'll discuss next.



1. Lambda job runs every 5 minutes
2. Request new filings from Edgar
3. Parse results and enqueue identifier for later processing

The first Lambda method is kicked off by a CloudWatch scheduled event which is like Cron for AWS. It runs every few minutes and which runs the Lambda function whose job is to look for new filings and queue them up for processing by the second Lambda method we'll discuss next.

Lambda 2: Data Processing

AWS Lambda

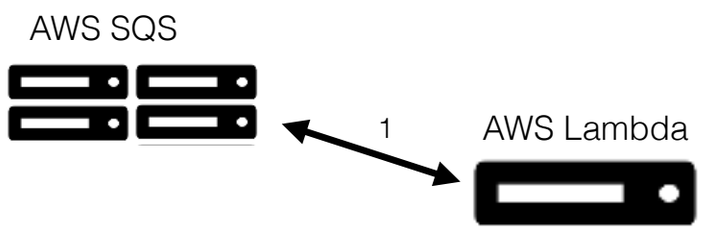
CODE

4. Processing Lambda jobs run via cron and pull items from queue for processing
5. Resulting data sent to ElasticSearch, files to S3
6. Remove items from queue

The second Lambda method can run X number of jobs also on a schedule whose job is to pull Edgar document IDs from an SQS queue and process each individually.

Lambda job runs every 5 minutes
Request new filings from Edgar
Parse results and enqueue identifier for later processing
Processing Lambda jobs run via cron and pull items from queue for processing
Resulting data sent to ElasticSearch, files to S3
Remove items from queue
Errors logged and developers alerted

Lambda 2: Data Processing

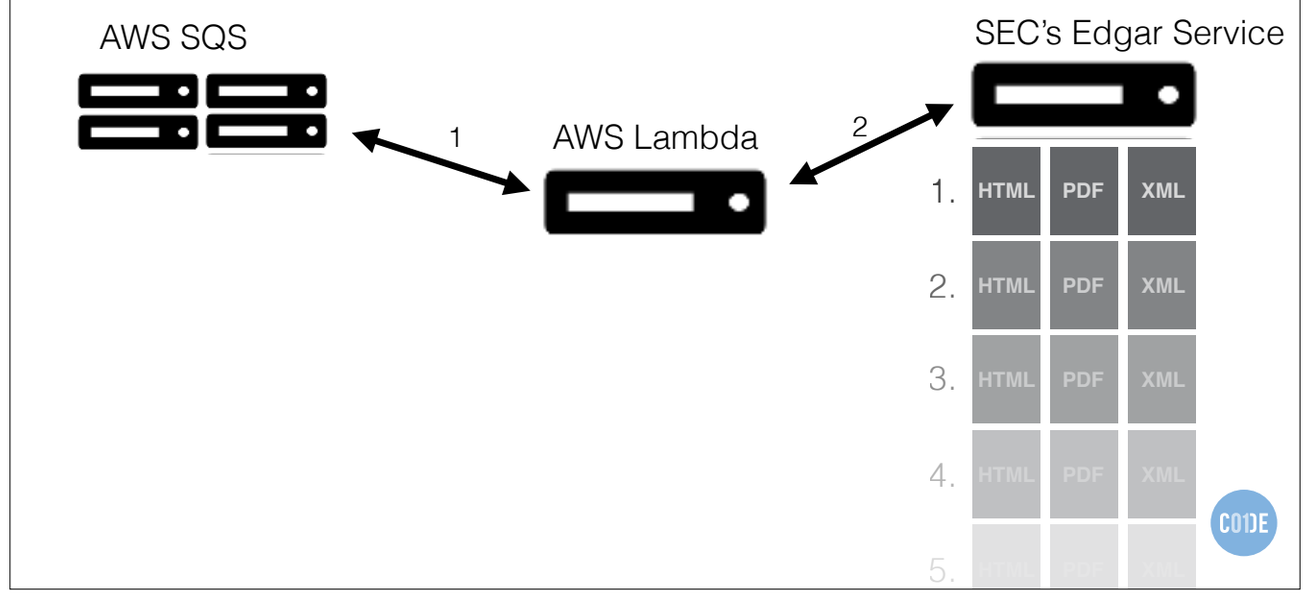


4. Processing Lambda jobs run via cron and pull items from queue for processing
5. Resulting data sent to ElasticSearch, files to S3
6. Remove items from queue

The second Lambda method can run X number of jobs also on a schedule whose job is to pull Edgar document IDs from an SQS queue and process each individually.

Lambda job runs every 5 minutes
Request new filings from Edgar
Parse results and enqueue identifier for later processing
Processing Lambda jobs run via cron and pull items from queue for processing
Resulting data sent to ElasticSearch, files to S3
Remove items from queue
Errors logged and developers alerted

Lambda 2: Data Processing

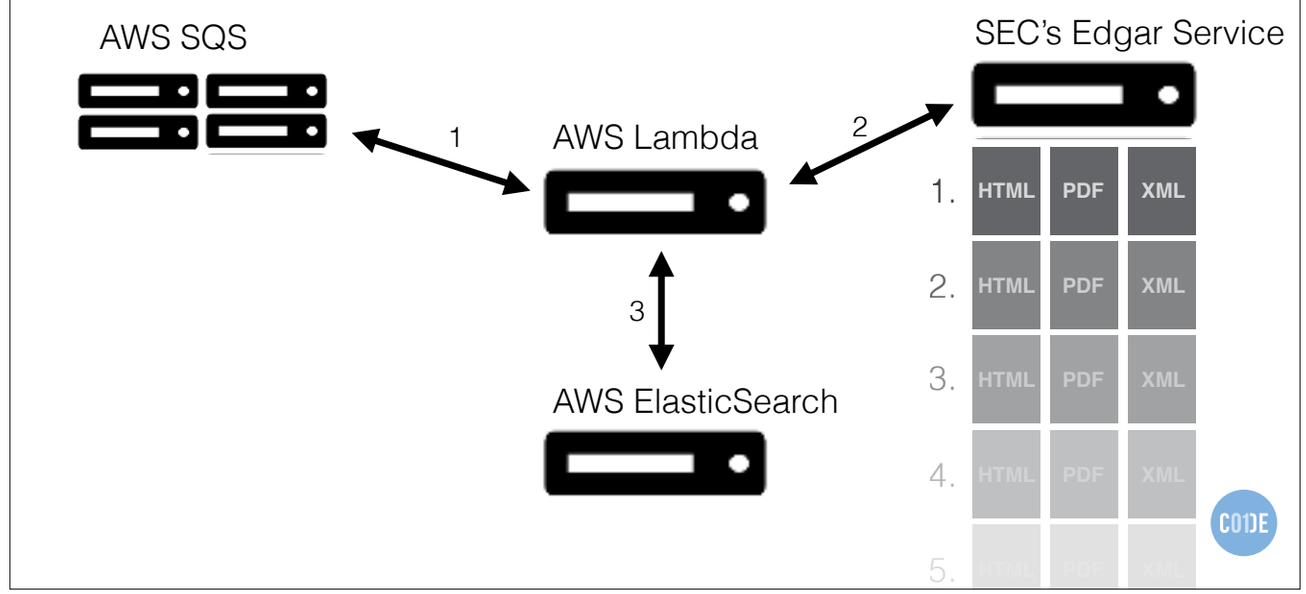


4. Processing Lambda jobs run via cron and pull items from queue for processing
5. Resulting data sent to ElasticSearch, files to S3
6. Remove items from queue

The second Lambda method can run X number of jobs also on a schedule whose job is to pull Edgar document IDs from an SQS queue and process each individually.

Lambda job runs every 5 minutes
Request new filings from Edgar
Parse results and enqueue identifier for later processing
Processing Lambda jobs run via cron and pull items from queue for processing
Resulting data sent to ElasticSearch, files to S3
Remove items from queue
Errors logged and developers alerted

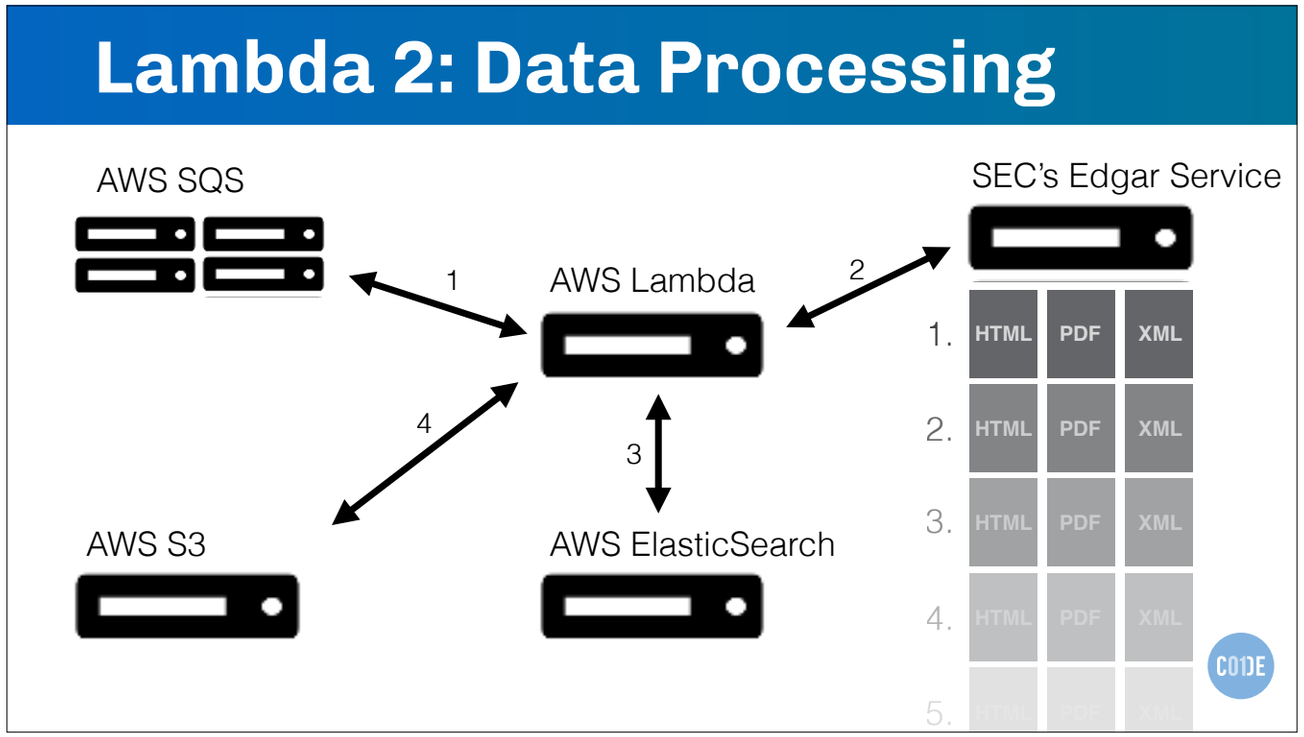
Lambda 2: Data Processing



4. Processing Lambda jobs run via cron and pull items from queue for processing
5. Resulting data sent to ElasticSearch, files to S3
6. Remove items from queue

The second Lambda method can run X number of jobs also on a schedule whose job is to pull Edgar document IDs from an SQS queue and process each individually.

Lambda job runs every 5 minutes
Request new filings from Edgar
Parse results and enqueue identifier for later processing
Processing Lambda jobs run via cron and pull items from queue for processing
Resulting data sent to ElasticSearch, files to S3
Remove items from queue
Errors logged and developers alerted



4. Processing Lambda jobs run via cron and pull items from queue for processing
5. Resulting data sent to ElasticSearch, files to S3
6. Remove items from queue

The second Lambda method can run X number of jobs also on a schedule whose job is to pull Edgar document IDs from an SQS queue and process each individually. This is what we did for importing historical information.

The Website

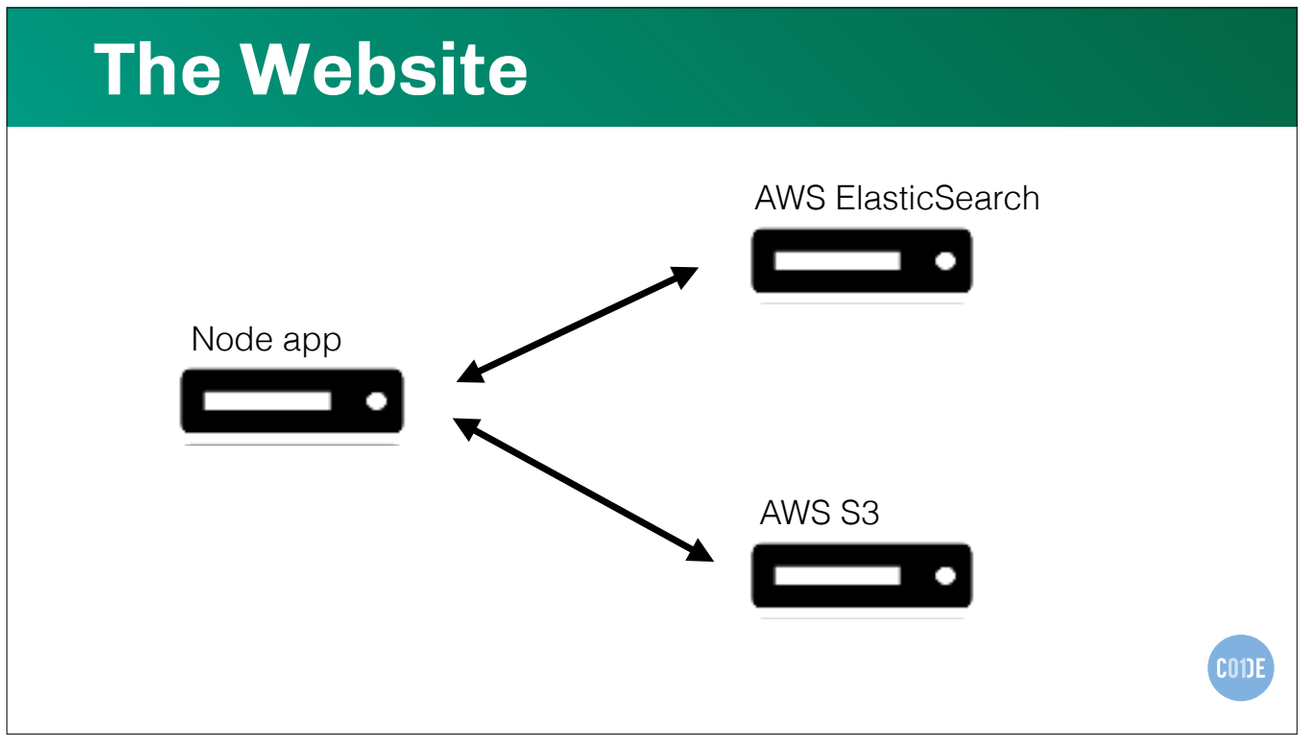
Since we had an API to request data from, we were basically constructing queries, requesting data, and rendering it to a page.

We could use any language that could pull data from ElasticSearch and render html. There was no persistence on the server and it could be rebuilt at any time. It had a much less responsibility than our monolithic Rails application would have afforded.

No matter how we process data from Edgar, it does not effect our website traffic. And if our website popularity explodes, it does not negatively impact the data processing.

Think about: complicated setup and management puts (possible) undo burden on client and their team. How can you limit the maintenance required? What needs to be managed for the continued health of your application?

Request data from ElasticSearch
Render HTML pages



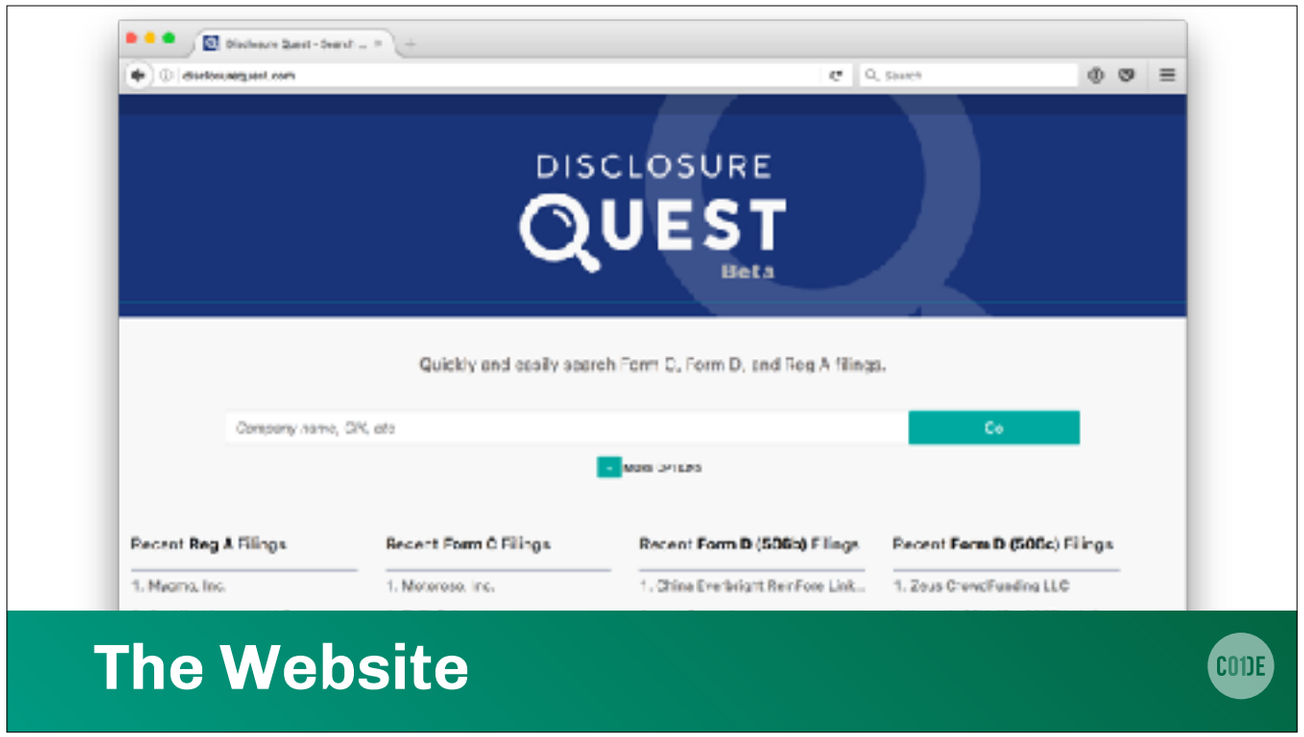
Since we had an API to request data from, we were basically constructing queries, requesting data, and rendering it to a page.

We could use any language that could pull data from ElasticSearch and render html. There was no persistence on the server and it could be rebuilt at any time. It had a much less responsibility than our monolithic Rails application would have afforded.

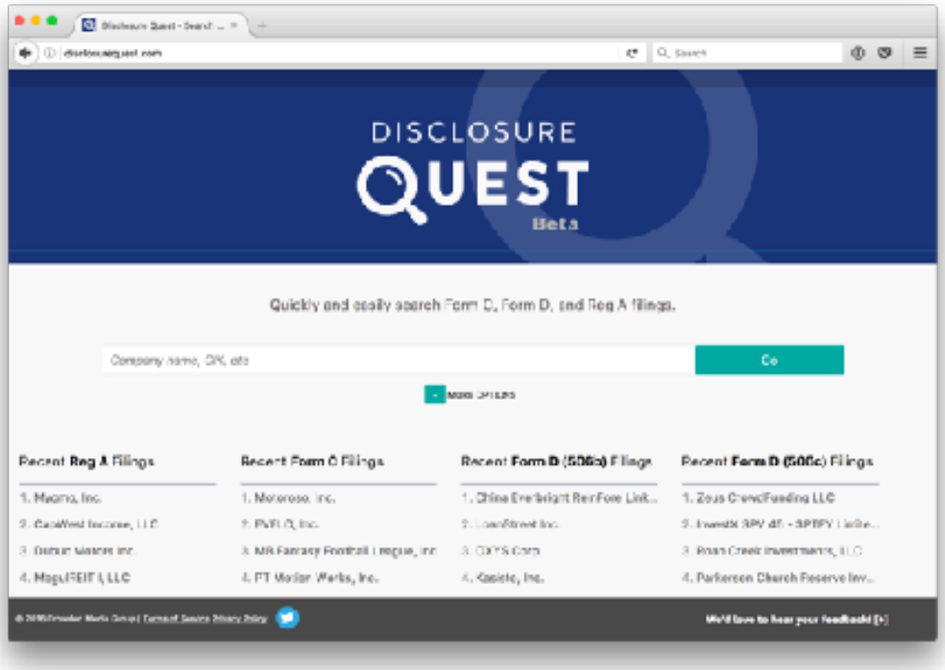
No matter how we process data from Edgar, it does not effect our website traffic. And if our website popularity explodes, it does not negatively impact the data processing.

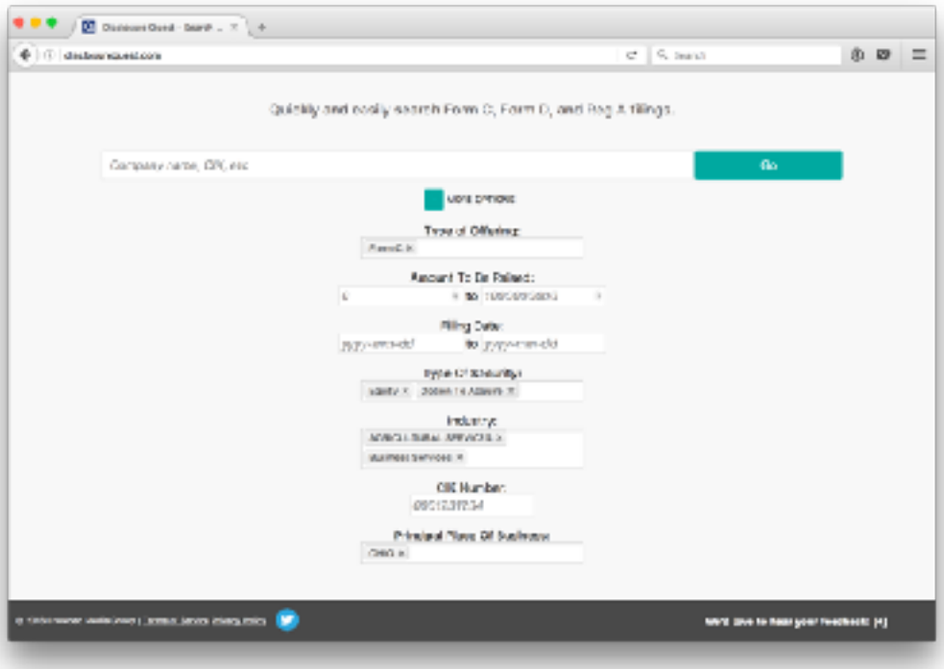
Think about: complicated setup and management puts (possible) undo burden on client and their team. How can you limit the maintenance required? What needs to be managed for the continued health of your application?

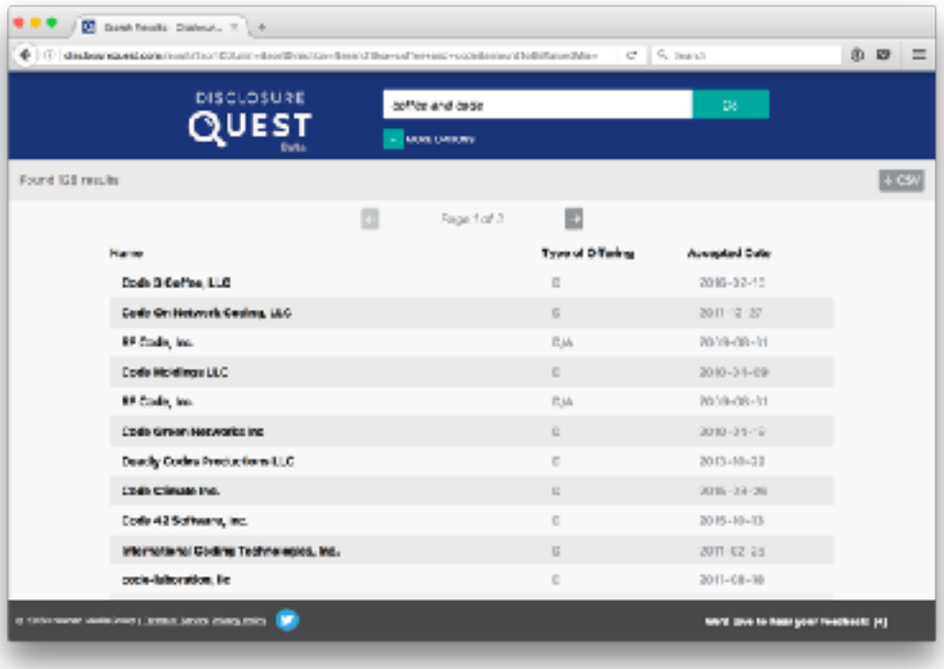
Request data from ElasticSearch
Render HTML pages

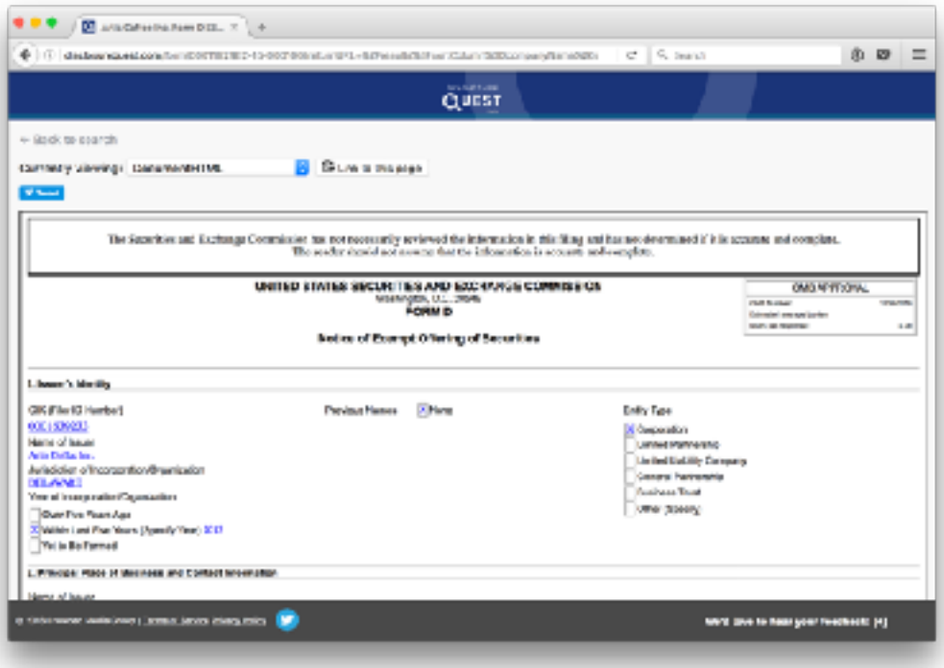

DISCLOSURE
QUEST
Beta
Quickly and easily search Form C, Form D, and Reg A filings.
Company name, CIK, etc
Go
Recent Reg A Filings
Recent Form C Filings
Recent Form D (506b) Filings
Recent Form D (506c) Filings

The Website


DISCLOSURE
QUEST
Beta
Quickly and easily search Form C, Form D, and Reg A filings.
Company name, CIK, etc
Go
More Options
Recent Reg A Filings
1. Means, Inc.
Recent Form C Filings
1. Monrosa, Inc.
Recent Form D (506b) Filings
Recent Form D (506c) Filings
1. Zeus CrowdFunding LLC
4. MagLREIT I, LLC
4. PT Motor Works, Inc.
4. Kaskela, Inc.
We'd love to hear your feedback [>]


Quickly and easily search Form C, Form D, and Reg A filings.
Company name, CIK, etc
Go
More Options
Type of Offering:
Amount To Be Raised:
Filing Date:
Industry:
CIK Number:
Principal Place Of Business:


DISCLOSURE
QUEST
Found 108 results
+ CSV
Page 1 of 3
Name
Type of Filing
Accepted Date

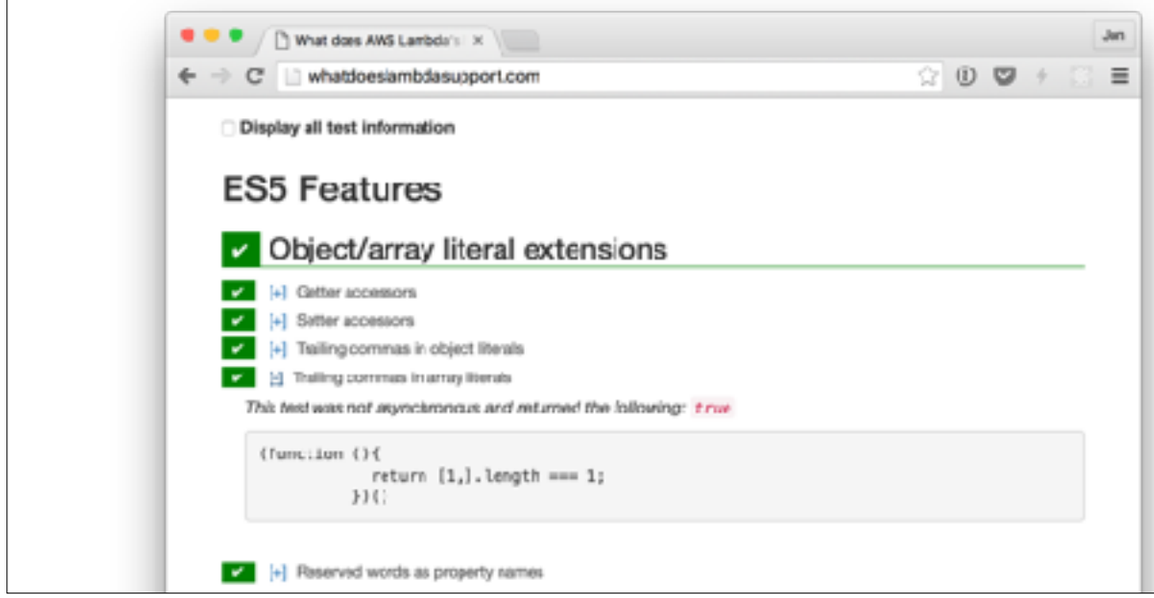


We met our goals for the Disclose Quest project and were able to deliver an application that was easy to maintain, could scale automatically, and was easy to understand. We also were able to make a better deliverable that aligned with our “hidden objectives”.

Other Project Examples

- API Gateway to handle HTML form submissions
- Create one-off API endpoints w/out bloating codebase
- Manage web hook responses / send alerts to Slack
- Recreate static website from dynamic data on schedule

whatdoeslambdasupport.com



CODE

Lambda runtime finally updated to recent-ish version of Node, need to know what it supports: whatdoeslambdasupport.com

Uses the Kangax test suite which shows support for ES5/6/7+ javascript features in the latest browser versions.


Tooling
CODE

Serverless (node)

- Most popular Node.js deployment library
- Working on a platform as a service
- Aspirations of running on other platforms

CODE

serverless is a Node application framework for creating Lambda functions and tying them to http endpoints via AWS API Gateway.

Apex (node/go/java/python)

- Built by TJ Holowaychuk of Node.js fame
- Uses Node.js shim to load Go code

CODE

AWS SAM (node/java/python)

- Simplified “application” YAML config
- Produced AWS CloudFormation file
- Uses Lambda, API Gateway, and DynamoDB
- Can utilize Swagger API definitions
- Requires multiple step deployment

https://aws.amazon.com/blogs/compute/introducing-simplified-serverless-application-deplyoment-and-management/

Honorable Mentions

Tooling

- Zappa
- Gordon

Providers

- iron.io
- Azure Functions
- Google CloudFunctions

Apex used Lambda and Go, Zappa uses Lambda and python, Gordon uses Lambda, python, and cloud formation.
iron.io as SaS with distributed job processing, distributed queue system, and a caching service.
Azure rolled out their own version of Lambda with event driven tasks that can be exposed as HTTP endpoints.

New and Shiny

- Environment variables
- Lambda dead letter queues
- API gateway routes with ANY and star routes
- Modify cloud front headers on requests at endpoint
- Upgrade of node.js environment (originally v0.10.4)
- AWS Serverless Application Model (SAM) file
- KMS integration to encrypt/decrypt env values

https://aws.amazon.com/blogs/compute/simplify-serverless-applications-with-environment-variables-in-aws-lambda/

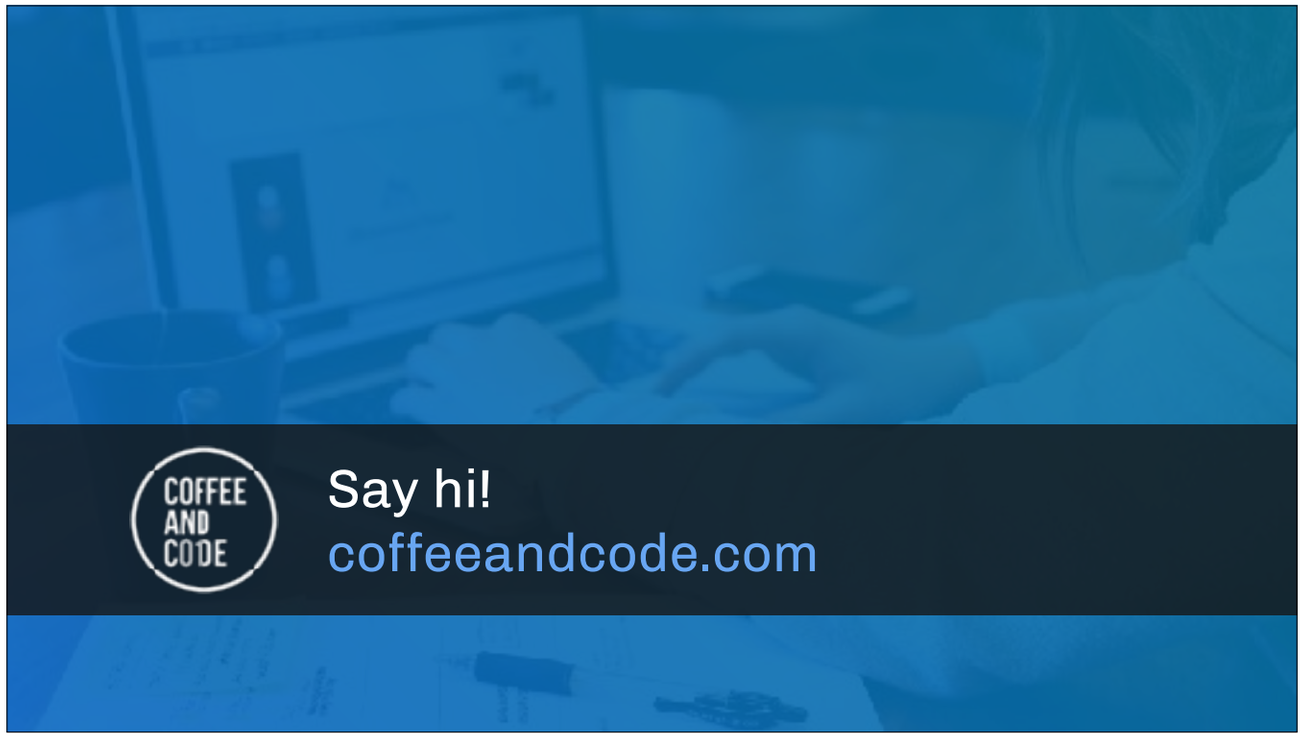
https://aws.amazon.com/blogs/compute/introducing-simplified-serverless-application-deplyoment-and-management/

Lessons Learned

- Log vigorously
- Use error notification service and/or CloudWatch alerts
- AWS API Gateway could be much better
- API Gateway works with Swagger definition files (kind of)
- Think of your application in distinct parts (microservices)
- Learn IAM roles and how to tie services together

CODE


Questions?
CODE



In closing, EventsXD for feedback or please stop by and let me know what you thought about the content.

That’s all I have for now on the subject, but please remember to think about how you can simplify the products you are delivering.

But if you need some assistance, you can give me a call. Thank you very much.