serverless-iiif:
from flail to scale in a
year or less

10/27/2020
Samvera Connect 2020

Michael Klein (Northwestern University Libraries)
Trey Pendragon (Princeton University Library)
Edward Silverton (Mnemoscene)
Jonathan Hartzler (University of Notre Dame)
Background

- 2016: Northwestern IT starts encouraging units to build/move applications and infrastructure to Amazon Web Services
- 2017: NUL’s Audiovisual Repository (built on Avalon Media System) moves from on-premise data center to AWS
- 2018: NUL has AVR and two Hyrax apps running in staging and production on AWS, maintained through Terraform
- Shared stack
  - Fedora, Solr, Zookeeper on Elastic Beanstalk
  - Postgresql on RDS
  - Elasticache Redis
  - S3 and Elastic File System storage
Cloud Architecture

- Services > Servers
- Availability > “Uptime”
- All instances are ephemeral
- Local storage is an illusion
- Never copy when you can stream
IIIF in the Cloud

- Access is bursty by nature
- Tile requests are at least an order of magnitude burstier
- Very hard to scale a server-based solution in a way that can handle the bursts without wasting capacity (and money)
- **Cantaloupe Image Server**
  - Flexible, fast, and powerful
  - Easy to deploy
  - Wants to copy and cache
  - Scales at the server level
  - Loads of great features, most of which we didn’t need
Solution

**liif-processor** – turns a IIIF URL path into an image or info.json result

**serverless-iiif** – Lambda function wrapper function that mediates between the web service and the node module

Converts web requests into AWS Lambda calls and returns the result to the client

**Image Storage**

**Application versioning, packaging, quick deployment, and reusability**
Under the Hood

- **libvips**
  - “A demand-driven, horizontally threaded image processing library”
  - Very fast and extremely lightweight
  - Can perform more than 300 different image manipulation operations
  - Supports a large number of image formats
  - Most importantly (for our purposes), it supports multi-resolution images and tiling

- **sharp**
  - Fast native Node.js wrapper for libvips
  - Supports seeking and streaming of image data without making a local copy
Deploying

serverless-iiif

Description
An IIIF 2.1 Image API compliant server written as an AWS Serverless Application.

Components
- A simple Lambda Function wrapper for the iiif-processor module.
- A Lambda Layer containing all the dependencies for the Lambda Function.
- An API Gateway interface for the Lambda Function.
- A CloudFormation template describing the resources needed to deploy the application.

Prerequisites
- Some basic knowledge of AWS.
- An Amazon Web Services account with permissions to create resources via the console and/or command line.
- An Amazon S3 bucket to hold the source images to be served via IIIF. Note: The Lambda Function will be granted read access to this bucket.

Quick Start

Application settings

Application name
The stack name of this application created via AWS CloudFormation

serverless-iiif

iiif.LambdaTimeout
The timeout for the lambda.

10

SourceBucket
Name of bucket containing source images

StageName
Name of the API stage to be deployed

latest

I acknowledge that this app creates custom IAM roles.

Deploy
Deploying

If you want to go far, go together!
Deploying
Metrics

If you want to go far, go together!
Royal Pavilion & Museums

• **Strapi.io CMS**
• Custom CSV import of TIFF files to S3 bucket
• Strapi API endpoint for IIIF manifests (also generic record requests)
• Lambda function running serverless-iiif for image tiles
• Customised to work in all AWS regions with Michael Klein
• < £10 p/m running costs (inc Heroku)
• 400% increase in traffic for “Close Look” blog
• Zero ongoing devops overhead (so far!)
• Wanted to try moving from Kakadu-powered Cantaloupe/Loris

• OpenJPEG was too slow, Cantaloupe would crash with any reasonable amount of traffic.

• Tried Pyramidal Tiffs + Cantaloupe, still had throughput issues. Too easy to saturate the server and crash it. Further, cleaning up caches was often a sudden surprise, lots of concern over space.
Performance

• Basic serverless-iiif worked fine!
• Tested it with Siege, found the lambdas would freeze up. :(  
  • Happened in Production too, I also may have broken Northwestern’s while testing.
• Found dynamically figuring out width/height was the problem, now we store it in the object metadata (this is in core now!), which solved our issue there!
• Had to shrink several of our images so they’d still zoom enough for micro-detail, but not be slow or break lambda’s output limit. Cut size in half if > 15k pixels.
• Added Cloudfront caching.
1. Converted everything in digital repository from JP2 to Pyramidal Tiffs (took a few weeks, manifests pointed to serverless-iiif as they were generated).
2. 14.2 TB across 4.76M images, serving 5.76M requests/month.
3. 40% of requests served by Cloudfront cache.
4. $370/month, $252 is S3 (Intelligent Tier, 99% in Infrequent Access), $73 is Lambda, $33 is Cloudfront, $12 is API Gateway.
5. Never afraid of running out of space, it doesn’t suddenly fall apart, it handles any scale, we don’t have to do any cleanup!
Princeton uses Datadog for metrics, having a Lambda made a dashboard easy to build.

You want the invocations to be as low as possible compared to the Cloudfront hits.
Notre Dame Library

- Started a project in 2018 to implement iiif at Notre Dame.
- We have campus mandates to put as much new infrastructure as we can into AWS.
- Initially went with Cantaloupe as the image server.
  - Quickly realized that we had a lot of scaling problems.
    - Traffic from iiif viewers is very spiky
    - Cantaloupe was crashing due to memory usage in the VM environment
  - The napkin cost estimates to have running virtual machines with enough power to serve moderate use was very high.
Discovered serverless-iiif just as we were starting to talk about writing a lambda to do this ourselves.

Results were instantly better.
- We stopped having crashes.
- Average response times dropped by a factor of 4.
Notre Dame Library

- Found an opportunity to contribute back
- Rewrote the wrapper function for speed and maintainability
- Added comprehensive test coverage
- Iterated at Notre Dame
- Submitted an omnibus pull request to MBK yesterday
- Brought us to version 2.0.1 just in time for this presentation
Current Limitations & Future Work

• Northwestern isn’t *really* using serverless-iiif! But we’d like to be!
  • It was just a prop for a lightning talk at Open Repositories 2019!
• Workaround for the 6 Megabyte limit on API Gateway Lambda integrations
• Support for reading scale factors and tile sizes out of the image
  • Current tile size is hardcoded at 512x512
  • Current scale factors are computed by starting at full size and dividing by 2 until width < 64px
• Authentication and Authorization