Issue 47, 2020-02-17

Scaling IIIF Image Tiling in the Cloud

The International Archive of Women in Architecture, established at Virginia Tech in 1985, collects books, biographical information, and published materials from nearly 40 countries that are divided into around 450 collections. In order to provide public access to these collections, we built an application using the IIIF APIs to pre-generate image tiles and manifests which are statically served in the AWS cloud. We established an automatic image processing pipeline using a suite of AWS services to implement microservices in Lambda and Docker. By doing so, we reduced the processing time for terabytes of images from weeks to days.

In this article, we describe our serverless architecture design and implementations, elaborate the technical solution on integrating multiple AWS services with other techniques into the application, and describe our streamlined and scalable approach to handle extremely large image datasets. Finally, we show the significantly improved performance compared to traditional processing architectures along with a cost evaluation.

by Yinlin Chen, Soumik Ghosh, Tingting Jiang, James Tuttle

Introduction

For over 30 years, Virginia Tech University Libraries' Special Collections and Archives has amassed a hidden treasure of architectural drawings, design sketches, personal and professional correspondence, project files, business records, photographs by more than 400 women who practiced architecture and design around the world from the 1800s through the present day. The collections,

ISSN 1940-5758

including many oversized materials, were only available on-premises until 2019.

In 2016, Virginia Tech University Libraries was awarded a Digitizing Hidden Collections grant from the Council on Library Resources (CLIR). The grant, *Women of Design: Revealing Women's Hidden Contributions to the Built Environment* [1], partially funded the scanning, description, and provision of access to 800 cubic feet of materials from the IAWA collections.

Through the generous support of CLIR, Virginia Tech University Libraries' Special Collections and Archives digitized and described a significant part of the collection. The grant also supported collaboration between Special Collections and Archives and the Libraries' Information Technology Services to design and build a web application to make the work of 30 women architects freely accessible to the world through an online image repository: International Archive of Women in Architecture (IAWA) [2].

To enhance access to Virginia Tech's cultural heritage collections, IAWA supports the International Image Interoperability Framework (IIIF) [3], providing a world-class image viewing and sharing experience by exposing high-quality digital images along with rich metadata. IIIF is a set of shared application programming interface (API) specifications built to support image interoperability across digital image repositories. Enriched by a community of software tools and technology systems, IIIF enables:

1. A uniform and rich access to image-based resources hosted around the globe by repositories promoting the common technical specifications
2. Better, faster and cheaper image delivery with IIIF compatible image servers and web clients
3. Advanced features for viewing and interacting with structured collections of images by IIIF API-enabled image viewers.

Due to limitations in our on-premise infrastructure and rapidly increasing computing and storage needs during this project, we suspected a cloud solution might best address our needs. Relatively early in the project, Libraries' Network Attached Storage was nearing capacity at over 90 percent utilization. Budget and time constraints conspired against expanding on-premise storage within the project timeline, which led us to use AWS Simple Storage Service (S3) [4], Amazon's cloud object storage, which was used for other purposes on campus. Numerous efficiencies were easily realized after moving the data to the AWS Cloud.

Hosting the data in AWS Cloud facilitated leveraging other AWS services. To improve performance and lower latency to and from our application in addition to reducing data transfer costs, we leverage Amazon CloudFront [5], a content delivery network (CDN) service, to deliver static content. To perform a long-running and compute-intensive task, we use AWS Batch [6], a service that manages batch computing workloads. Finally, to receive system event and response action notifications, process intermediate data, and trigger other AWS services, we use AWS Lambda [7] to implement our customized microservices. We describe in detail our implemented AWS solutions in the following sections.

Problem description

IAWA is a Hyrax/Savmvera application [8] supported by Fedora [9]. Specifically, it is based on Hyrax 2.1.0 and Fedora 4. Initially, we took the native, out-of-the-box approach, and ingested high-resolution TIFF images and stored them directly in Fedora. We integrated an IIIF-API enabled viewer – Mirador [10] – with frontend Hyrax to display our image collections. The viewer supports zoom, pan, and other features, as well as selection among related images.

The underlying image tile and manifest generation are processed in real-time. Upon receiving a user's request, the application first retrieves all the collection image files from Fedora, stores them into the server's temporary location, subsequently triggers the IIIF image service using the riiif gem [11] to start creating tiles and manifests. After the generation process is completed, the Mirador viewer can then use the generated IIIF URL to display the images to the end-user. This is the program logic as it was implemented in Hyrax [12]. However, this implementation is only suitable for a small amount of image data and the application performance drops significantly when both the size of image files and the number of images in the collection are large. In the IAWA project, for example, a collection might contain 20 pages (image files) at 100MB each. Under this circumstance, even with 32GB RAM and 50GB storage space allocated, the image viewing process became unbearably slow, often causing the application to halt and crash. There was an alternative implementation of caching the manifests to avoid frequent Fedora access; however, due to the large number of our image collections, the response time to newly requested images was still too long for a production service. Moreover, this approach added the overhead of periodically running a rake task to refresh and rebuild the cache. Thus, we sought an approach to display images from pre-generated image tiles and manifests instead of generating them on the fly in hopes of eliminating the need to make exhaustive search requests

in Fedora. Our solution was to build an IIIF level 0 compatible image server with all the pre-generated files in place. Since all the tiles and manifests are ready to serve the requests, the website performance increased immediately.

Our challenge then became how to efficiently generate terabytes of IAWA image collection metadata and tiles. We performed a local test that used a batch script to generate a set of image collection of around 20GB, which took about a week to finish. Extrapolating out the time it would take to process the expected size of the entire digitization process, several months or possibly a year, it was clear that this was not a viable approach. Finally, we designed and implemented our cloud-based AWS solution and successfully generated all our IAWA image collections in 3 days. We describe our architecture design, software implementation, deployment, and performance in the next sections.

System Architecture

This system architecture was designed with several goals in mind.

1. The system provides a single entrypoint to assign tasks.
2. It can run multiple tasks in parallel and adjust the resource needs for each task automatically without human intervention.
3. The system can monitor task status and produce detailed task and cost reports.
4. It reduces or eliminates maintenance needs by using managed services in a serverless architecture.

Figure 1 shows an overview of our system architecture design. We use Amazon S3 and AWS Batch management services and implement our business logic as microservices using AWS Lambda. After a user uploads a task file to a designated folder in S3, an AWS Lambda is triggered automatically and parses the content in the task file and then submits multiple jobs to AWS Batch. These batch jobs run in parallel and each job processes a set of image collections and uploads generated tiles and manifests into S3 buckets. When batch jobs are finished, AWS Batch disposes of resources that are no longer needed. Given this design, resources are provisioned only when needed to process tasks. An added benefit to AWS managed services is that there is no longer a need to perform system maintenance such as installing OS updates, security patches, etc.

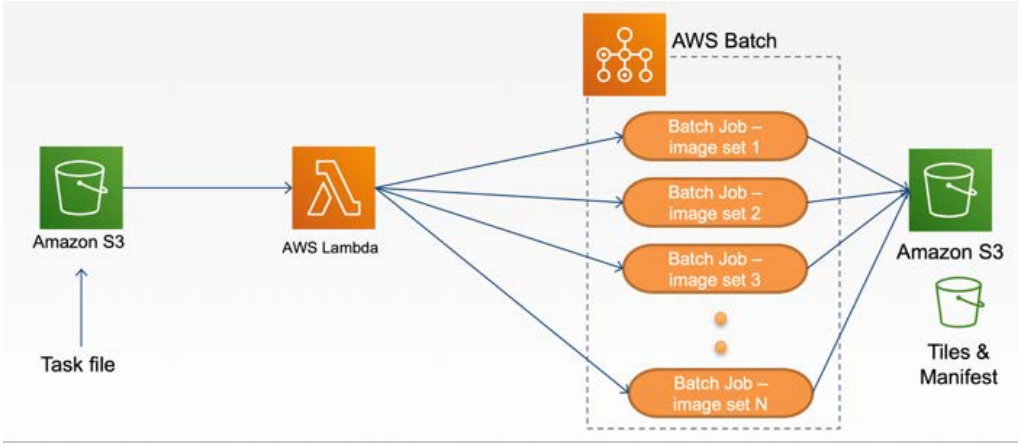

Figure 1. An overview of our AWS cloud-based image tiling system architecture

The IIIF tile and manifest generation relies on several AWS components. There is an S3 bucket for task files and another bucket for tile and manifest files. Our image collections reside in another S3 bucket. An AWS Batch is instantiated by creating a job definition, compute environment, and a job queue. A job definition specifies the type of job to be run and the amount of resources to be allocated. For example, it may specify a Docker image, the number of vCPUs, amount of memory, what command to be executed, etc. A compute environment defines what kind of EC2 instance is to be provisioned and used to run the AWS Batch jobs. A job queue is mapped to one or more compute environments and a job is submitted to the job queue. Once there are jobs in the job queue, AWS Batch will acquire the necessary resources, prepare the compute environment with Amazon EC2 instances, and schedule these jobs to be run within the instances. Then, our microservice implementation is deployed in the AWS Lambda. The entire system configuration with instructions is published in our project GitHub page. [13]

Software Implementation

Our software implementations include an AWS Lambda function, an IIIF Ruby gem implementation, and an executable script. We package the Ruby gem and executable script into a Docker image, which is the core business logic component to be run in the AWS Batch job. Figure 2 illustrates the overarching workflow inside the AWS Batch job.

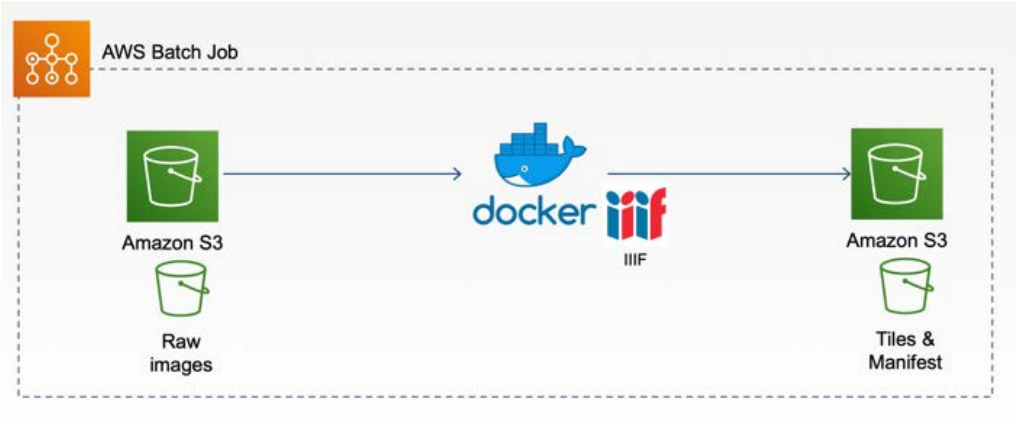

Figure 2. The overarching workflow inside our AWS Batch job

The IIIF Ruby gem generates two IIIF items. One is IIIF image API compatible image tiles that can be requested and delivered using IIIF specific URIs, and the other is image metadata and the structure/layout of the image objects can be displayed through IIIF presentation API compatible image manifests. With these two IIIF items, we can support advanced viewing capabilities, including deep zoom for individual images, metadata information sharing as well as structured/ordered view of images in a collection. Currently, there are several libraries capable of generating IIIF level 0-compatible image tiles and manifests for the image files that can be statically served in local or cloud image server, such as iiif_s3 [14]. However, the implementation of an IIIF image viewer, such as Mirador, which we are using, and Universal Viewer require specific image metadata information in order to properly display the images with the above viewing capabilities. Although IIIF specifications support bare thumbnails as string objects, Mirador only accepts thumbnails as image API service objects. Thus, we extended the iiif_s3 work and implemented our own IIIF Ruby gem giiif [15] to fulfill this specific requirement. Furthermore, we fixed a bug in iiif_s3 when using ImageMagic to transform high-resolution image files to default (e.g., JPEG) format: if the original image is in the default format, the transformation step should be skipped in order to avoid a code break.

The executable script is a set of bash commands to execute the workflow. It first fetches the images to be processed into a local disk inside the Docker container. It then executes a Ruby script [16] which gathers the required information for generating tiles and manifests, including metadata CSV file for image collections, the path to the folder containing image files in the collections, and root path information for the generated image stack. Finally, it uploads all the generated files (tiles and manifests) to the designated location in S3.

To automate the IIIF tiling process, we implemented an AWS Lambda

function that processes task definition files uploaded to an S3 folder and submits AWS Batch jobs. When a task file is added to a specific folder in S3, this Lambda function automatically triggered. The Lambda function reads the content in the task file, including parameters such as job definition, job queue, command to execute, environment parameters list, etc and then submits an AWS Batch job respectively. Figure 3 shows the Python code snippet representing this procedure.

```
jobName = content['jobName']
jobQueue = content['jobQueue']
jobDefinition = content['jobDefinition']
command = []
command.append(content['command'])
wait = None

envlist = createEnvList(content['environment'])

submitJobResponse = batch.submit_job(
    jobName=jobName,
    jobQueue=jobQueue,
    jobDefinition=jobDefinition,
    containerOverrides={
        'command': command,
        'environment': envlist
    }
)
```

Figure 3. AWS Lambda function in Python: receives task and submits AWS Batch job

Finally, we update our IAWA Hyrax application by storing an item's IIIF manifest URL in a new metadata field called import_url. This IIIF manifest URL contains links to the pre-generated tiles for that item. When a user views the images in an item using the Mirador viewer, the application does not query Fedora for image data; it instead retrieves the information from S3. More specifically, the program now reads a static string instead of calling a complex and time-consuming function to retrieve the image metadata and create the tiles on the fly. With this change, users can view an image in milliseconds to seconds, compared to minutes in the original implementation. The performance improvement is approximately 1,000x faster.

Figure 4 is the code snippet shows the difference between our implementation (Left) and Hyrax implementation (Right)

```
23  def manifest_url
24    import_url
25  end
```

```
174  def manifest_url
175    manifest_helper.polymorphic_url([:manifest, self])
176  end
```

Figure 4. IAWA (left) vs. Hyrax (right) implementations for image manifest

Deployments and Tasks Executions

As mentioned briefly in the System Architecture section, AWS Batch organizes its work into four components:

- Jobs – the unit of work submitted to AWS Batch.
- Job Definition – defines how jobs are to be run, memory and CPU requirements, Docker container properties, environment variables, and mount points for persistent storage
- Job Queues – submitted jobs go into queues with priorities. Queues can be associated with one or more compute environments.
- Compute Environment – the compute resources that are used to run jobs.

When jobs are scheduled to run, they are placed into a single queue associated with a compute environment. At a single point in time, there can be multiple active queues trickling jobs into their respective compute environments. We can see all job statuses, number of jobs in each queue, and compute environment information through AWS Batch Dashboard, which is shown in Figure 5.

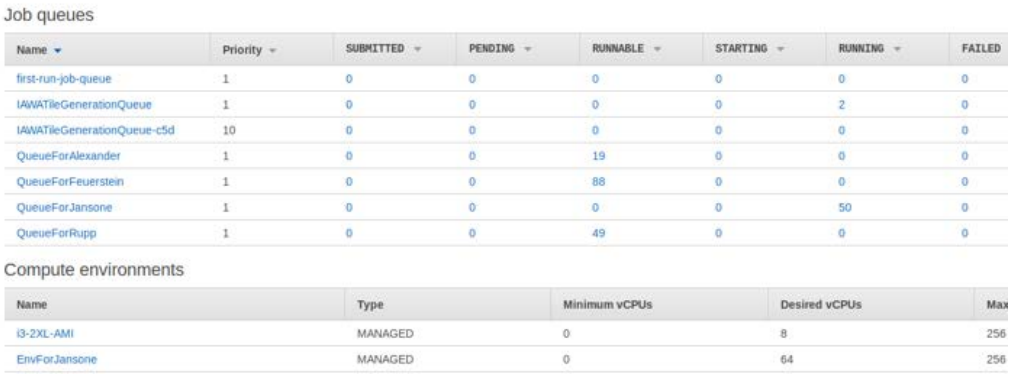

Figure 5. AWS Batch Dashboard

We can see more granular information, such as the logs from running Docker containers through Cloudwatch as shown in Figure 6. Cloudwatch logs are updated in real-time as well as retained after job completion.

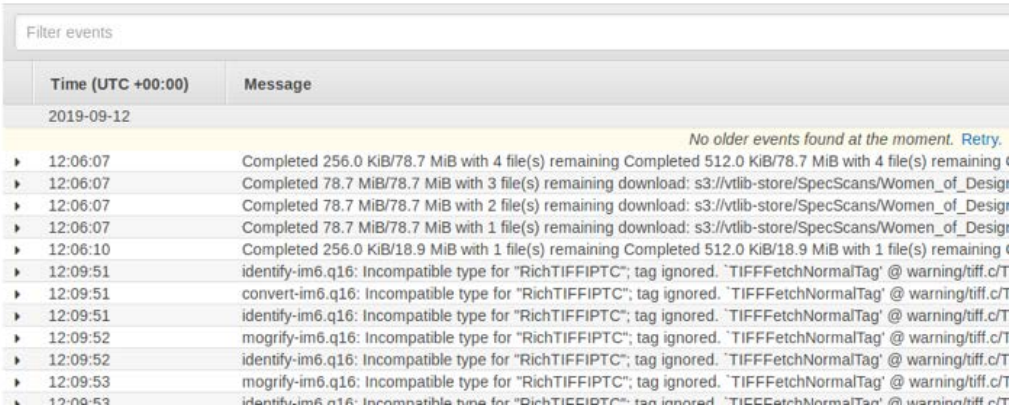

Figure 6. Cloudwatch log showing the output of a running job

Performance and Cost

There are 31 IAWA image collections and sizes ranged from tens of gigabytes to hundreds. We selected four collections to conduct the performance and cost for this article. The smallest one we tagged for calculating costs was around 23 GB while the largest one was upwards of 230 GB. Table 1 shows a breakdown of the execution time for these collections.

Table 1. Execution time of the selected collections

Collection Name	Size (GB)	# of tasks	Time
Ms1995_007_Piomelli	229.80 GB	62	12 hours
Ms2007_007_Feuerstein	149.43 GB	44	5 hours
Ms2008_089_Alexander	35.14 GB	19	4 hours
Ms2016_012_Womens_Development_Corp	22.99 GB	16	3 hours

The total cost of generating tiles for these collections was $36.54. The smallest collection cost $1.71 while the largest one cost $20.08. As we can see, this method of generating tiles by leveraging the concurrency of cloud services is fast and extremely cost efficient.

Table 2. Cost to process selected collection

Collection Name	Size (GB)	Cost
Ms1995_007_Piomelli	229.80 GB	$20.08

Ms2007_007_Feuerstein	149.43 GB	$12.93
Ms2008_089_Alexander	35.14 GB	$1.82
Ms2016_012_Womens_Development_Corp	22.99 GB	$1.71
Total	437.36 GB	$36.54

IAWA Application

The IIIF tile generation system is deployed in AWS and we currently use this system to process all IAWA image collections. Whenever a new IAWA collection with raw image files along with the metadata CSV files are ready to be processed, SysOps need only create a task file for the new collection and upload that task file to S3 at which point the automated process will initiate. Figure 7 shows one page of image files in an item (a group of image files). With our new design, no matter how large each image size or how many images that item contains, our application can display it with a fast and stable response time.

Figure 7. Item page with multiple images

Conclusion

In this article, we describe how we establish an auto-scaling system to generate IIIF tiles using AWS services. Our proposed serverless architecture design eliminates the need to manage underlying servers and delegate all the heavy lifting, e.g. auto-scaling, instance provisioning, scheduling, resource deprovisioning, etc to AWS. We focus on implementing microservices and meshing AWS to accomplish our tasks. This system can process any amount of image collections automatically and efficiently. We reduce the time to process a collection from weeks to hours and simplify the human process from several manual operations to a simple file upload step. We also share our AWS experience and implementations details about the entire system building and deployment process. Finally, we publish our work in the authors' institution GitHub page [13].

Future work

This proposed system is deployed in the AWS and becomes one of our cloud-based image processing services. While we manually deployed this system into AWS currently, we start working on developing an AWS CloudFormation and/or Terraform template in order to have the ability to deploy the entire system at one click of a button. Furthermore, our proposed serverless architecture can be extended to perform other tasks by simply replacing the software in the Docker image and changing the AWS Batch configuration files. For example, we can package JHOVE software [17] into a Docker image and configure the AWS Batch to use this new Docker image. A new service is then established and ready to perform format identification, validation, and characterization of digital objects. We will generalize this procedure and use this approach to create other Library services.

References

[1] Women of Design [Internet]. [updated 2019]. Available from: http://registry.clir.org/projects/23081249

[2] International Archive of Women in Architecture (IAWA) [Internet]. [updated 2019 August 1]. Blacksburg (VA): University Libraries Special Collections and Information Technology department, Virginia Tech. Available from: https://iawa.lib.vt.edu/

[3] International Image Interoperability Framework (IIIF) [Internet]. [updated 2019 July 1]. Available from: https://iiif.io/about/

[4] Amazon Simple Storage Service (Amazon S3) [Internet]. [updated 2019]. Available from: https://aws.amazon.com/s3/

[5] Amazon CloudFront [Internet]. [updated 2019]. Available from: https://aws.amazon.com/cloudfront/

[6] AWS Batch [Internet]. [updated 2019]. Available from: https://aws.amazon.com/batch/

[7] AWS Lambda [Internet]. [updated 2019]. Available from: https://aws.amazon.com/lambda/

[8] Hyrax/Samvera [Internet]. [updated 2019]. Available from: https://hyrax.samvera.org/

[9] Fedora [Internet]. [updated 2019]. Available from: https://duraspace.org/fedora/

[10] Mirador Viewer: [Internet]. [updated 2019 August 10]. Available from: https://projectmirador.org/

[11] riiif Ruby gem: [Internet]. [updated 2018 April 30]. Available from: https://github.com/curationexperts/riiif

[12] Hyrax GitHub: [Internet]. [updated 2018 March 30]. Available from: https://github.com/samvera/hyrax/blob/v2.1.0/app/presenters/hyrax/displays_image.rb#L15

[13] AWS Batch IIIF generator: [Internet] [updated 2019]. Available from: https://github.com/VTUL/aws-batch-iiif-generator

[14] iiif_s3: A generator for IIIF level 0 compatible static server on Amazon S3. [Internet] [updated 2017 November 26]. Available from: https://rubygems.org/gems/iiif_s3/versions/0.1.0

[15] giif: IIIF compatible image stack and metadata generator. [Internet] [updated 2019 August 12]. Available from: https://github.com/VTUL/giiif

[16] A Ruby script to gather image-related information, generate tiles and upload to Amazon S3. [Internet] [updated 2019 August 12]. Available from: https://github.com/VTUL/image-iiif-s3/blob/master/create_iiif_s3.rb

[17] JHOVE: A file format identification, validation and characterisation tool. [Internet] [updated 2019 April 18]. Available from: https://jhove.openpreservation.org/

About the Authors

Yinlin Chen (ylchen@vt.edu) is a Digital Library Architect at the Virginia Tech Libraries, Blacksburg, VA. He holds a Ph.D. in Computer Science and Applications from Virginia Tech and a M.S. and B.S. in Computer

Science at National Tsing Hua University, Taiwan. His professional interests include digital libraries, cloud computing, microservices and serverless architecture.

Soumik Ghosh (soumikgh@vt.edu) is a Systems Engineer at Virginia Tech Libraries since 2017. He received his Masters in Computer Science and Applications from Virginia Tech in 2017 and his Bachelors in Computer Science from West Bengal University of Technology. Apart from containerisation and cloud native architectures, he likes to dabble in casual reading, writing, astronomy and microbiology.

Tingting Jiang (virjtt03@vt.edu) has been a Software Engineer at Virginia Tech Libraries since 2014. She received the B.S. (summa cum laude) and M.S. degrees in Computer Science from Virginia Tech, Blacksburg, VA. From 2007 to 2009, she was a Software Engineer with Intrexon Corporation, Blacksburg, VA. She was a recipient of an NSF Graduate Research Fellowship (2011–2014) and a Microsoft Research Graduate Women's Scholarship (2011).

James Tuttle (james.tuttle@vt.edu) is Associate Director for Digital Libraries at the Virginia Tech Libraries, Blacksburg, VA. He received the MSLIS and BA, Anthropology from the University of Illinois at Urbana-Champaign. He specializes in digital curation and preservation of complex materials, systems design, workflow analysis and efficiency design, and general systems planning with experience in higher education and industry.