Meadow: An Introduction To Northwestern University's New Digital Repository Application Built With Elixir, React, and GraphQL In The Middle

(https://github.com/nulib/meadow)

Brendan Quinn
Senior Developer
Northwestern University Libraries

Team Members

- Adam Arling: front end senior developer
- Divya Katpally: front end developer
- Michael Klein: lead developer
- Brendan Quinn: senior developer
- Veronica Robinson: service owner (welcome to the team!)
- David Schober: team lead and project/product manager
- Karen Shaw: senior developer and scrum lead

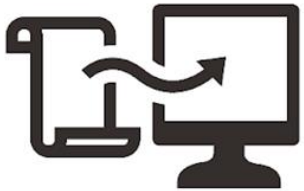
Special shout out to Laura Alagna, our Digital Preservation Librarian, who recently moved to a new position outside the library. We wish you the best!

Northwestern

Project Goals

A New Workflow: Minimizing the Danger Zone

ITEM DIGITIZED → **ITEM INGESTED TO PRESERVATION STORAGE** → **ITEMS DESCRIBED VIA BATCH ACTIONS** → **ITEMS PUBLISHED**



After works are inventoried, they are **digitized at scale** using in-house equipment or vendors depending on the complexity and rarity of the object.

Files are uploaded to an ingest staging bucket in **Amazon S3**.

Digitized works are recorded on an **Ingest Sheet** (a spreadsheet listing every file, accession number, and role) and QC'd.

Using a modified **Ingest Sheet**, the files are ingested and organized by work with minimal administrative metadata applied en-masse.

Files are placed in preservation storage (**S3/Glacier** synced to a local enterprise solution) and ***critical preservation tasks are executed***.

Audit trail created for preservation/file-level actions. Derivatives such as **pyramidal TIFs** are generated.

Using the UI or a batch **export / edit / re-import** workflow, **works** are described by metadata and organized into user-facing collections.

Metadata is stored in **Postgres** and written to **Elasticsearch**.

Files remain at rest in preservation storage, all actions take place without affecting preserved file.

Work metadata is QC'd and items are published (visibility is set).

Published **works** become accessible via separate Digital Collections (**React**) front end and the **ElasticsearchAPI**.

Project Goals

- Preservation-first design
- Fast and scalable ingest pipeline
- Ability to batch edit metadata
- Concurrent
- Fault-tolerant
- “Nice” development environment

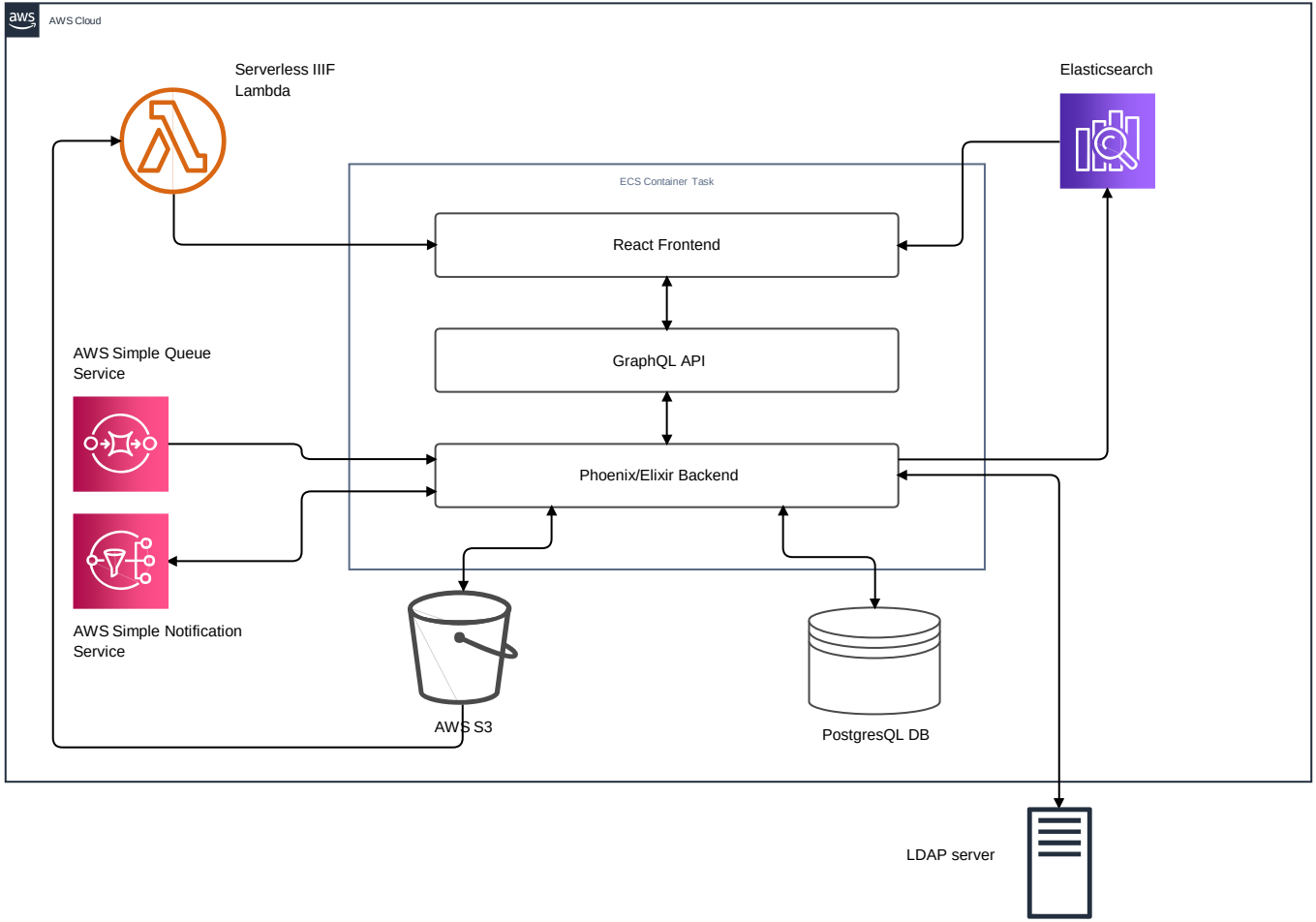
Architecture Overview

Architecture Overview

- Elixir backend: https://elixir-lang.org/
  - Phoenix Framework: https://www.phoenixframework.org/
- Javascript frontend
  - React: https://reactjs.org/
  - ReactiveSearch: https://opensource.appbase.io/reactivesearch/
- GraphQL API
  - Absinthe: https://absinthe-graphql.org/
- Search engine
  - Elasticsearch: https://www.elastic.co/elasticsearch/
- Amazon Web Services: https://aws.amazon.com/
  - API Gateway, ECS, AWS Lambda, S3, SNS, SQS, (& more alphabet soup…)
- Docker containerization: https://www.docker.com/

Architecture Overview

Elixir + Phoenix
|> GraphQL + Absinthe
|> Javascript + React


AWS Cloud
Serverless IIIF
Lambda
Elasticsearch
ECS Container Task
React Frontend
GraphQL API
Phoenix/Elixir Backend
AWS Simple Queue
Service
AWS Simple Notification
Service
AWS S3
PostgresQL DB
LDAP server

Metadata + Data = “MeadowData”

Data Model

The challenge

- 50+ descriptive and administrative properties on Works
- 15+ properties backed by external authorities or local controlled lists of terms
- Complex UI components and backend functionality needed to be built out to support this amount of information in logical and understandable ways to end users.

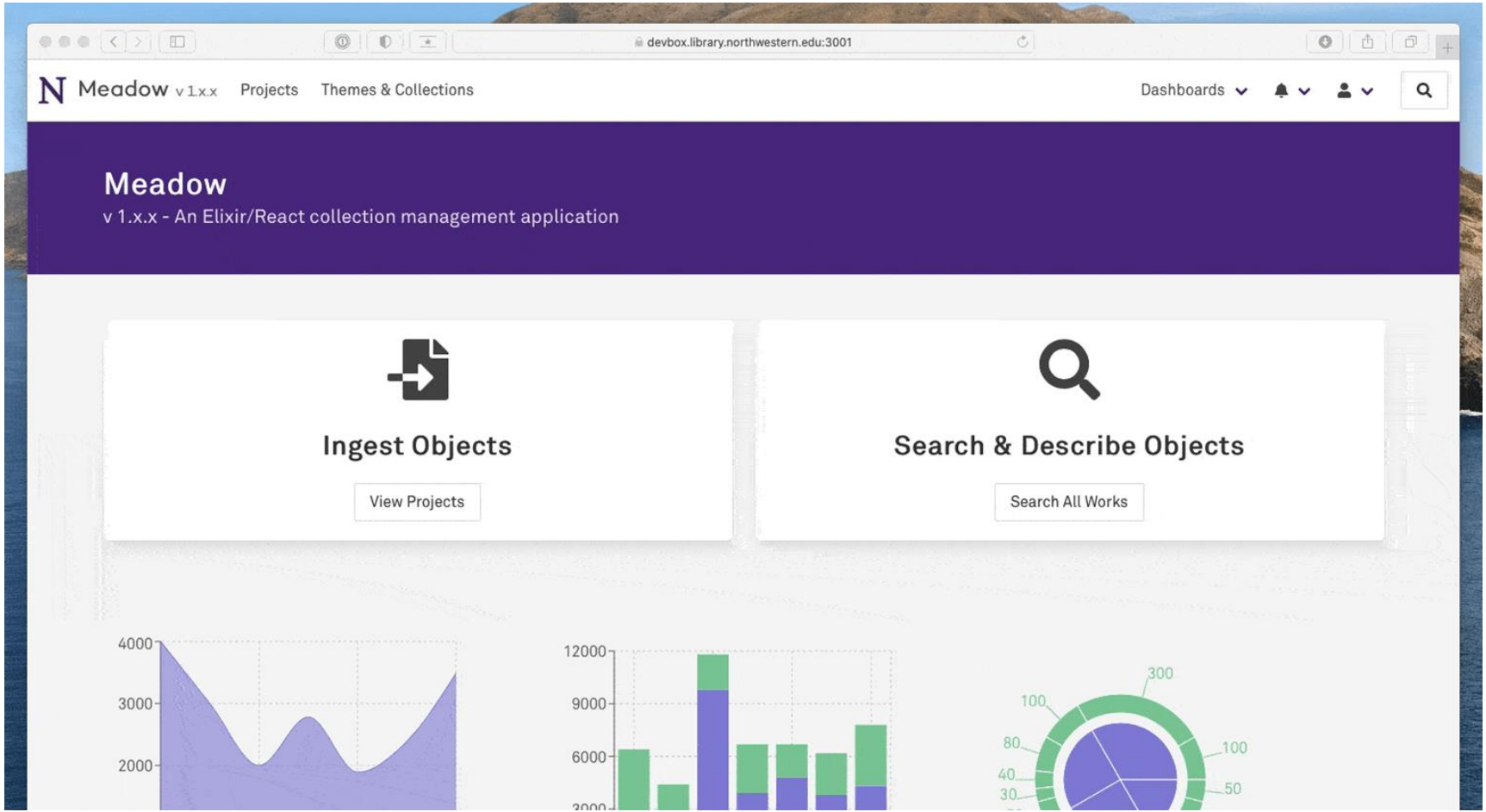
Data Model

The implementation

- Driven by **Ecto**, Elixir's most prominent library for data mapping and database interaction.
- We're using the PostgreSQL implementation of the **ecto_sql** library (you can use Ecto without any databases if you just need changesets for data mapping or validation).
- Relationships between Work, FileSet, & Collections schemas implemented using relational data functionality provided by Ecto ("has_many", "belongs_to", etc.)
- We make use of Ecto's embedded schemas to store Administrative and Descriptive metadata as jsonb directly on Work and FileSet records.
- Ecto queries and changesets underpin our GraphQL API for building out frontend queries and mutations.

Batch Ingest

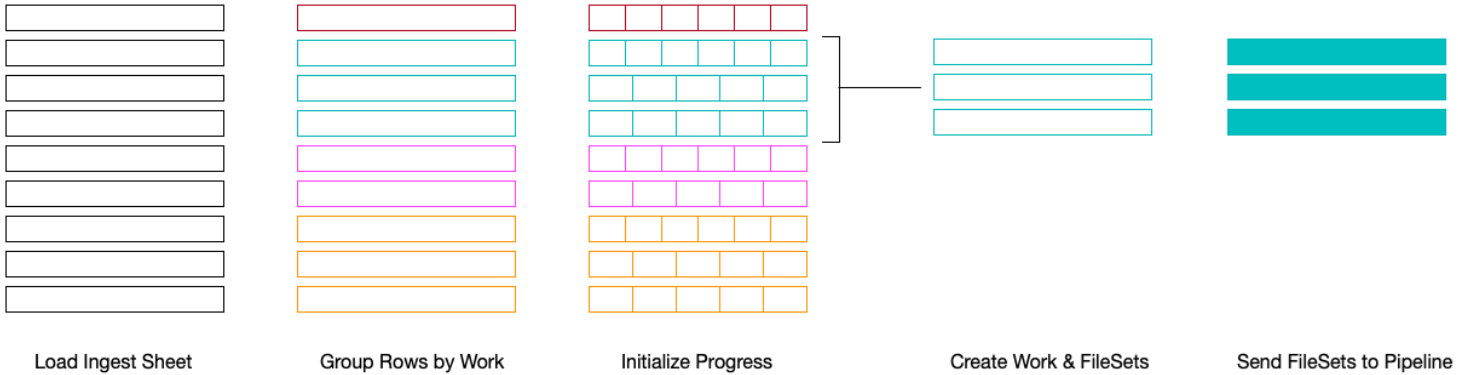
Previously Recorded Demo Slide!


devbox.library.northwestern.edu:3001
Meadow v 1.x.x
Projects
Themes & Collections
Dashboards
Meadow
v 1.x.x - An Elixir/React collection management application
Ingest Objects
View Projects
Search & Describe Objects
Search All Works
4000
3000
2000
12000
9000
6000
300
100
80
100
40
50
30

Ingest Sheet Processing

work_accession_number	accession_number	filename	description	role
work_acc_001	work_acc_001_01	inu-wint-61.3.jpg	Winterton Collection 1	am
work_acc_001	work_acc_001_02	inu-wint-64-9.jpg	Winterton Collection 2	am
work_acc_001	work_acc_001_03	inu-wint-65-22.jpg	Winterton Collection 3	am
work_acc_001	work_acc_001_04	inu-wint-66-11.jpg	Winterton Collection 4	am
work_acc_002	work_acc_002_01	inu-wint-66-20.jpg	Winterton Collection 5	pm
work_acc_002	work_acc_002_02	coffee.jpg	Coffee!	pm

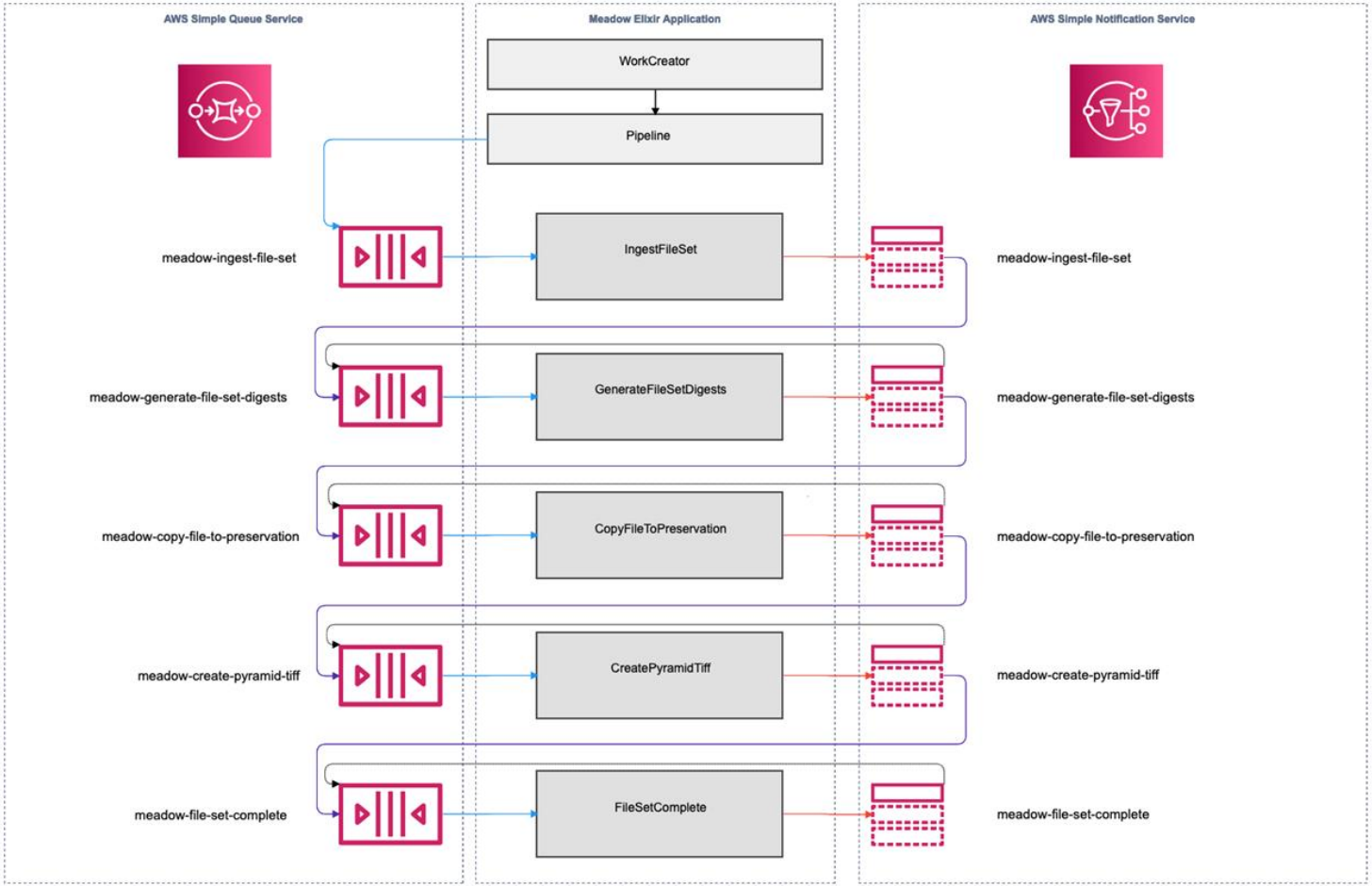
Ingest Spreadsheet Processing



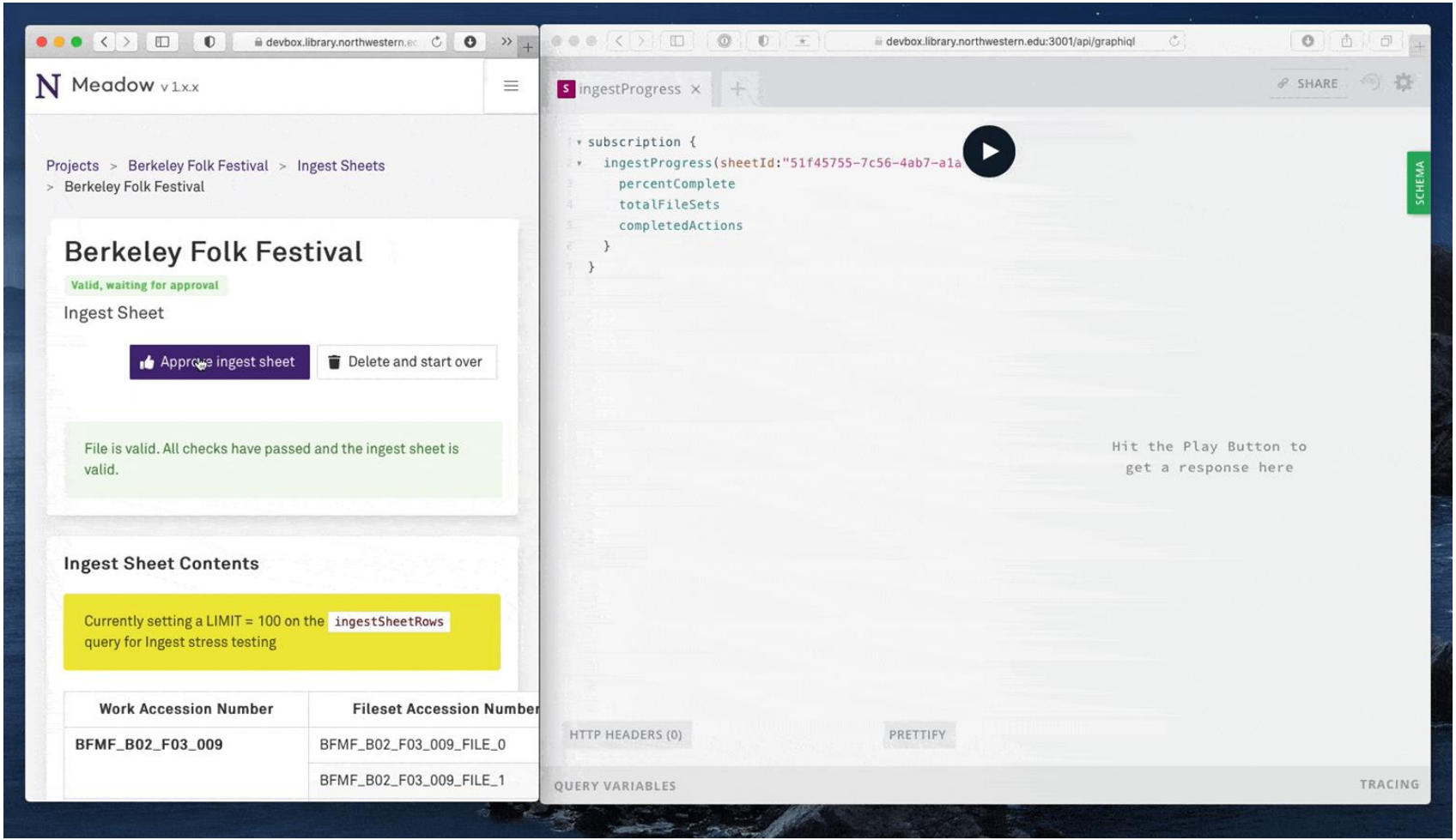
Sequins & Broadway

Sequins wraps the processing pipeline of an Elixir library called **Broadway SQS** to allow for the simple creation of multi-stage “SQS -> Broadway -> SNS” pipelines.

- Provides utilities to create the queues, topics, and subscriptions required to support the processing pipeline.
- Sets up pipeline infrastructure based on a list of queue/topic/subscription specifications
- Defines a standard behaviour for pipeline actions
  - only need to define a “process” function for each action
  - must return a status (:ok, :retry, :error) with optional data (in our case, a file_set_id)
- Allows flexible configuration options for concurrency
- Is a supervised Elixir application with built-in fault tolerance and error recovery. If a process crashes for any reason, it can be recovered back to a known “good” state.


AWS Simple Queue Service
Meadow Elixir Application
AWS Simple Notification Service
WorkCreator
Pipeline
meadow-ingest-file-set
IngestFileSet
meadow-ingest-file-set
meadow-generate-file-set-digests
GenerateFileSetDigests
meadow-generate-file-set-digests
meadow-copy-file-to-preservation
CopyFileToPreservation
meadow-copy-file-to-preservation
meadow-create-pyramid-tiff
CreatePyramidTiff
meadow-create-pyramid-tiff
meadow-file-set-complete
FileSetComplete
meadow-file-set-complete

Another Previously Recorded Demo Slide!


devbox.library.northwestern.ec
Meadow v 1.x.x
Projects > Berkeley Folk Festival > Ingest Sheets
> Berkeley Folk Festival
Berkeley Folk Festival
Valid, waiting for approval
Ingest Sheet
Approve ingest sheet
Delete and start over
File is valid. All checks have passed and the ingest sheet is valid.
Ingest Sheet Contents
Currently setting a LIMIT = 100 on the ingestSheetRows query for Ingest stress testing
Work Accession Number
Fileset Accession Number
BFMF_B02_F03_009
BFMF_B02_F03_009_FILE_0
BFMF_B02_F03_009_FILE_1
devbox.library.northwestern.edu:3001/api/graphiql
ingestProgress
SHARE
subscription {
ingestProgress(sheetId:"51f45755-7c56-4ab7-a1a
percentComplete
totalFileSets
completedActions
}
}
SCHEMA
Hit the Play Button to get a response here
HTTP HEADERS (0)
PRETTIFY
QUERY VARIABLES
TRACING

TODO