ECCO Dataset Production

Greg Moore (greg.moore@jpl.nasa.gov)

Ian Fenty (ian.fenty@jpl.nasa.gov)

ECCO Annual Meeting

UT Austin Oden Institute for Computational Engineering and Sciences

March 20-22, 2024

Challenge:

Rapid Production of ECCO results in native and lat/lon NetCDF format for cloud-based general distribution via PO.DAAC

Goals:

- Dataset production automation to support real-time and quick-look distribution in local-, super-, and cloud-compute domains
- General applicability to all MITgcm-based models (regional models, custom grids, etc.)
- Simplified deployment to support wider audience:
  - Standard Python distribution: git clone + pip install
  - Docker/Singularity images

Background:

- ECCO Central State Estimate compute considerations (V4r4/V4r5, LLC 90):
  - Input variable type counts:
    - 131 standard physical model, 40 biogeochemical model
  - Output file type counts:
    - 131+40 native, 40 lat/lon
  - Frequency:
    - Daily mean, monthly mean for 32 years
  - Total file counts:
    - Input:
      - ~ 2.0 M files (171 variables * 365 days/year * 32 years)
      - 16.4 TB total input (15.9 TB daily mean + 0.5 TB monthly mean)
    - Output:
      - ~ 2.5 M files ((171 native + 40 latlon) * 365 days/year * 32 years)
      - 22 TB native + latlon
  - Rough i/o estimates:
    - 3D field files are ~50x larger than 2D field files
    - 63 of 131 standard physical model fields and 36 of 40 BGC are 3D
    - ~(100/170) 3D files * 2.0M * 1 sec/rw = ~ 650 hours

Background, cont:

- For V5, LLC 270:
  - File counts remain the same, but V5 file sizes are ~9x larger:
    - ~ 6000 i/o hours
    - ~ 150 TB input
    - ~ 200 TB output

- Efficient, flexible approach for dealing with current, expected compute loads and distribution formats is essential

Current effort builds on prior work:

- “V 1.0”:
    - ECCOv4-py (https://github.com/ECCO-GROUP/ECCOv4-py) (Forget, Fenty)
    - Core functionality for generating native and latlon publication-ready NetCDF4 files
    - Funded by NASA ACCESS Program in 2017

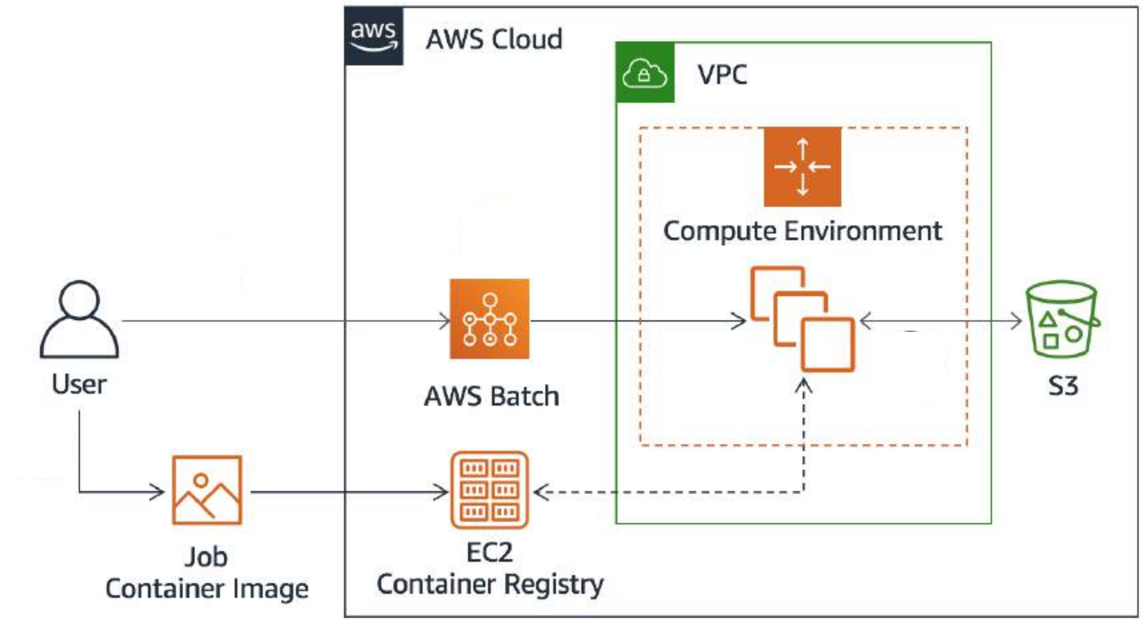
- “V 2.0”:
    - ECCO-Dataset-Production (https://github.com/ECCO-GROUP/ECCO-Dataset-Production) (Duncan Bark)
    - Migrated V 1.0 to AWS
        - Data storage in AWS S3
        - Parallel job submittal via AWS Lambda (“serverless compute”)
    - Limitations:
        - AWS Lambda instance limitations (10GB memory, 10GB container image, 15 minute function timeout)
        - S3 sync overhead
        - Manual config/install limited portability

Current effort: “V 3.0”

- Addresses V 2.0 limitations, anticipates future compute requirements:
  - Code has been reconfigured to support:
    - Standard Python setuptools-based deployment
      - ECCO utilities included as git submodules, package imports
      - Virtual environment (python –m venv <env>) or base installation
    - Containerized distribution
    - ”app”-oriented usage (mapping factor generation, AWS S3 sync, job submittal, etc.)

Current effort "V 3.0", cont:

- AWS Cloud implementation:
  - Batch/Fargate/ECS to overcome Lambda limitations:
    - Batch – queuing, scheduling, provisioning, compute instance management
    - Fargate – container-based "serverless" solution
    - ECS – container orchestration
    - S3 – object store



Current effort: “V 3.0”, cont.

- Status:
    - Completed:
        - Python packaging
        - Parallel AWS S3 sync and 2D/3D mapping factor applications
        - Docker container deployment via AWS ECR
        - AWS authorization/certificate abstraction to run in/outside of JPL domain (e.g., free tier accounts)
    - In progress:
        - Abstraction of data store/fetch for AWS/non-AWS environments
        - Batch/Fargate orchestration deployment
    - Expected release:
        - ASAP this quarter!!!