GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

HPS

Seminar Report

Continuous Integration For OCR-D In HPC

Abdallah Abdelnaby

MatrNr: 19334664

Supervisor: Giorgi Mamulashvili

Georg-August-Universität Göttingen
Institute of Computer Science

August 20, 2024

Abstract

The integration of DevOps approaches with High-Performance Computing (HPC) environments for Optical Character Recognition Development (OCR-D) workflows provides considerable improvements in efficiency, repeatability, collaboration, and scalability. This research shows how using CI/CD pipelines, containerization, and automation may improve traditional manual OCR procedures. Key findings include a significant reduction in processing times on HPC compared to ordinary PCs, especially for bigger datasets, demonstrating the efficiency benefits possible with HPC capabilities. Standard PCs, on the other hand, are more efficient for minor jobs (less than 35 pages) due to HPC systems' scheduling overhead. Automation using Infrastructure as Code (IaC) enables uniform and repeatable environments, which improves reliability. The use of centralized version control and automated workflows promotes cooperation, and containerization using Singularity allows for scalability across several HPC resources. This research demonstrates the revolutionary power of merging current software development processes with high-performance computing, while also providing a solid foundation for future scientific computing advancements.

Declaration on the use of ChatGPT and comparable tools in the context of examinations

In this work I have used ChatGPT or another AI as follows:

- ☐ Not at all
- ☑ During brainstorming
- ☐ When creating the outline
- ☐ To write individual passages, altogether to the extent of 0% of the entire text
- ☐ For the development of software source texts
- ☐ For optimizing or restructuring software source texts
- ☐ For proofreading or optimizing
- ☐ Further, namely: -

I hereby declare that I have stated all uses completely.
Missing or incorrect information will be considered as an attempt to cheat.

Contents

List of Tables

List of Figures

List of Listings

List of Abbreviations

HPC High-Performance Computing

OCR-D Optical Character Recognition Development

DevOps Development Operations

IaC Infrastructure as Code

CI/CD Continuous Integration / Continuous Delivery

CPU Central Processing Unit

GB Gigabyte(s)

XML Extensible Markup Language

GWDG Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen

1 Introduction

In recent years, the convergence of D (DevOps) methods with High-Performance Computing (HPC) has emerged as a viable strategy for improving the efficiency and reliability of scientific computing. DevOps, a combination of "development" and "operations," refers to a culture shift and set of practices designed to improve cooperation between software development and IT operations teams [Nuy23]. DevOps' main ideas include Continuous Integration/Continuous Delivery (CI/CD), Infrastructure as Code (I (IaC)), automation, and monitoring, all of which lead to shorter development cycles, better collaboration, and greater system stability [SHH18] [RMF20].

HPC environments are designed to deal with large-scale computations and big datasets, and are commonly utilized in sectors such as scientific research, engineering, and data analysis [Nuy23]. Historically, these settings depended on manual procedures and custom workflows, which were time-consuming and prone to mistakes [Nuy23] [SHH18] [RMF20]. HPC systems that use DevOps approaches might benefit from automated processes, uniform infrastructure, and short deployment cycles [Cou23]. This project investigates the integration of DevOps methods into HPC, with a special focus on operating OCR-D processes. OCR-D is an optical character recognition (OCR) software for digitizing historical documents and manuscripts [OCR24]. The objective is to establish a seamless CI/CD pipeline that automates the execution of OCR-D workflows on HPC equipment resulting to a simple and faster document digitization.

The project's goals are to build a continuous integration system for OCR-D in an HPC environment and to make it easier to execute OCR-D workflows using an automated CI/CD pipeline. The pipeline is designed to handle a variety of steps, including connecting to the HPC environment and fetching OCR-D Docker images, as well as building Singularity images, uploading data and scripts, sending OCR jobs, getting results, and executing cleanup chores.

By adopting this DevOps methodology in an HPC environment, the project hopes to illustrate the benefits of automation, repeatability, and efficiency in scientific computing. The incorporation of OCR-D processes demonstrates how contemporary software development approaches may be used to improve the capabilities and performance of HPC systems.

1.1 Motivation and Objectives

1.1.1 Motivation

The motivation for this project arises from the need to update and optimize scientific computing procedures, particularly in the field of Optical Character Recognition (OCR) in Digital Humanities. Traditional HPC systems, while powerful, sometimes rely on inefficient and error-prone manual operations. This can result in inefficiencies, uneven findings, and challenges in cooperation and reproducibility. By incorporating DevOps methods into HPC, we may overcome these issues and achieve various benefits.

1. Efficiency and speed: DevOps approaches like Continuous Integration/Continuous Delivery (CI/CD) may greatly accelerate development and deployment cycles, ensuring that updates and enhancements are quickly integrated and distributed.

2. Reproducibility: Automation and Infrastructure as Code (IaC) make environments more consistent and repeatable, lowering the chance of errors and making it easier to duplicate findings across multiple systems.

3. Collaboration: Centralized version control and automated processes improve cooperation between researchers, developers, and operators, allowing them to operate smoothly together.

4. Reliability and Consistency: Automated testing, monitoring, and deployment pipelines improve software and workflow reliability by assuring consistent performance under various situations.

5. Scalability: Containerization solutions such as Singularity allow us to efficiently scale applications across diverse HPC resources, maximizing the most of available computational power.

1.1.2 Objectives

The major goal of this project is to provide a continuous integration and delivery mechanism for executing OCR-D processes on HPC equipment. Specifically, the initiative intends to:

1. Create a CI/CD Pipeline: Using GitLab and GitLab Runner, automate the various phases of OCR-D process execution on HPC computers.

2. Implement Containerization: Use Singularity to containerize the OCR-D environment, ensuring portability and consistency across several HPC nodes.

3. Automate Data and Workflow Management: Automate the process of uploading data, workflows, and required scripts to the HPC environment, decreasing manual intervention and error.

4. Streamline Job Submission: Create scripts to automate the submission of OCR jobs to the HPC scheduler, ensuring that jobs are managed effectively and resources are used appropriately.

5. Retrieve and Process Results: Automate the retrieval of OCR job results and use technologies such as LAREX to improve the readability and usability of OCR output.

6. Optimize Resource Utilization: Investigate strategies for increasing processing speed via parallelization and the usage of proxy servers, as well as optimizing the pipeline to manage several concurrent operations.

7. Improve cooperation and Reproducibility: Make certain that the whole workflow, from code and data management to task execution and result processing, is managed in a way that encourages cooperation and reproducibility among researchers and developers.

By attaining these goals, the project hopes to demonstrate the efficacy of incorporating DevOps methods into HPC systems, hence delivering a solid, efficient, and scalable solution for OCR-D operations. This connection not only enhances the overall efficiency and reliability of OCR activities but also paves the way for modernizing other scientific computing procedures using DevOps approaches.

2 Methodology

The process for incorporating DevOps approaches into High-Performance Computing (HPC) to perform OCR-D processes consists of many critical phases and components. This section describes the procedures, technologies, and tools utilized to meet the project's goals, resulting in a streamlined and automated workflow for OCR jobs on HPC equipment.

2.1 Project Architecture

Figure 1 shows a CI/CD (Continuous Integration/Continuous Deployment) architecture that uses GitLab in combination with (G (GWDG)) (HPC) environment. The GitLab repository is at the center of this system, storing critical components such as batch scripts, workflows, data, results, and the.gitlab-ci.yml configuration file. This repository is essential for managing the project's source code and CI/CD setups.

The GitLab Runner in the GÖNET Nework executes CI/CD jobs described in the.gitlab-ci.yml file. It connects to the HPC environment via SSH, ensuring that the most recent code and pipeline definitions are pulled from the GitLab repository. The HPC environment itself consists of several components. The front end hosts, gwdu101, gwdu102, and gwdu103, are the principal access points for HPC resources, managing user logins and task submissions.

The SLURM scheduler is crucial to job management in the HPC environment, since it distributes resources and assigns jobs to the proper computer nodes where the computational operations are performed. The HPC configuration has two file systems: HOME, which stores user files and scripts permanently, and SCRATCH, which stores intermediate data and results temporarily. NextFlow, a workflow management system, is used to create and run sophisticated data processing pipelines, which communicate with the SLURM scheduler to efficiently manage activities across HPC resources.

A node with the hostname transfer-mdc.hpc.gwdg.de enables efficient data transfer between the HPC environment and external systems, which is crucial for operations that need huge datasets. The OCR-D Docker image, which includes the OCR-D (Optical Character Recognition - D) software environment, is converted into an OCR-D Singularity SIF (Singularity Image Format) for use in an HPC environment. Singularity is chosen over Docker in HPC environments because to its compatibility and performance benefits.

In this design, the workflow begins with a developer pushing changes to the GitLab repository. The GitLab Runner detects these changes and initiates the CI/CD pipeline, which connects to the HPC front-end hosts via SSH. Batch scripts and workflows from the repository are then sent to the SLURM scheduler, which assigns resources and runs the tasks on the computer nodes. NextFlow orchestrates processes, using the OCR-D Singularity SIF for specific tasks, and stores intermediate and final outcomes in the SCRATCH and HOME filesystems. Large datasets are transported through the data transfer node when needed, resulting in efficient data handling throughout the process.

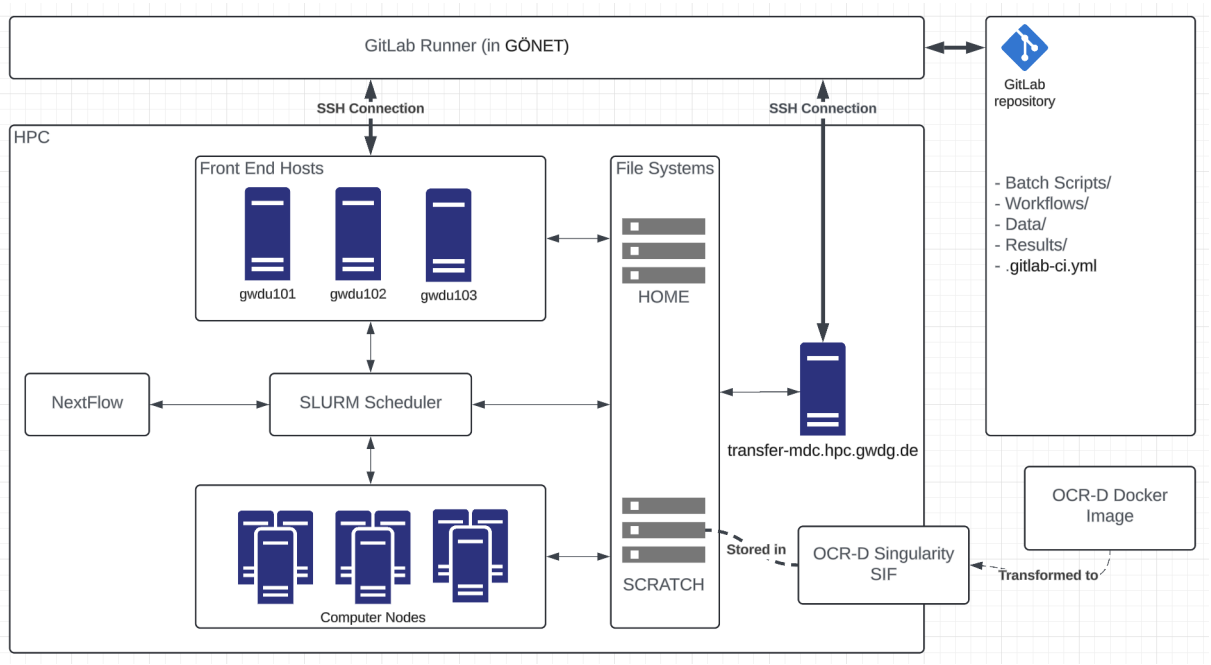


Figure 1: Project Architecture.

2.2 Project Setup

2.2.1 Environment Preparation

- **Singularity Containerization**: Create a Singularity Image File (SIF) from the OCR-D Docker image using a batch script (`batch_create_ocrd_all_maximum_sif.sh`). Singularity is chosen for its compatibility with HPC environments.
- **OCR-D Models**: Download necessary OCR-D models using a script (`batch_download_ocrd_all_models.sh`).

2.2.2 GitLab Repository

- **Repository Structure**: Organize the repository to include scripts, workflows, and data directories. This structure ensures that all components required for the OCR-D workflows are version-controlled and easily accessible.
- **CI/CD Configuration**: Set up the `.gitlab-ci.yml` file to define the CI/CD pipeline stages.

2.2.3 GitLab Runner

- **Configuration**: The GitLab Runner has to be configured in a network that could access the HPC data transfer endpoint

2.3 CI/CD Pipeline Stages

2.3.1 SSH Connection Setup

Listing 1 Ensures SSH keys are securely set up and available as environment variables in GitLab. This stage establishes a connection to the HPC environment.

```
setup_ssh:
  stage: setup_ssh
  script:
    - mkdir -p ~/.ssh
    - echo "$SSH_PRIVATE_KEY" | tr -d '\r' > ~/.ssh/id_rsa
    - chmod 600 ~/.ssh/id_rsa
    - echo "$SSH_KNOWN_HOSTS" > ~/.ssh/known_hosts
```

Listing 1: "Setup SSH Connection" in YAML

2.3.2 Upload Data and Workflows to HPC

Listing 2 Uses `upload_to_hpc.sh` to transfer data, workflows, and batch scripts to the HPC environment through the HPC transfer endpoint.

```
upload_to_hpc:
  stage: upload_to_hpc
  needs:
    - setup_ssh
  only:
    - tags
  script:
    - |
      WORKSPACE_NAME=${CI_COMMIT_REF_NAME}
      bash scripts/upload_to_hpc.sh $WORKSPACE_NAME $SCRATCH_BASE
```

Listing 2: "Upload to HPC" in YAML

2.3.3 Singularity Image Creation

Listing 3 Schedules the batch script to create the Singularity SIF from the OCR-D Docker image using `sbatch`.

```
create_singularity_image:
  stage: setup
  needs:
    - upload_to_hpc
  only:
    - tags
  script:
    - sbatch $SCRATCH_BASE/$WORKSPACE_NAME/scripts/batch_create_ocrd_all_maximum_sif.
```

Listing 3: "Create Singularity Image" in YAML

2.3.4 Download OCR-D Models

Listing 4 Schedules the batch script to download the OCR-D models using `sbatch`.

```yaml
download_ocrd_models:
  stage: download_models
  needs:
    - setup_environment
  only:
    - tags
  script:
    - sbatch $SCRATCH_BASE/$WORKSPACE_NAME/scripts/batch_download_ocrd_all_models.sh
```

Listing 4: "Download OCR-D Models" in YAML

2.3.5 Submit Workflow Job

Listing 5 Schedules the batch script to submit the OCR job to the HPC scheduler using `sbatch`.

```yaml
submit_ocrd_job:
  stage: submit_job
  needs:
    - download_ocrd_models
  only:
    - tags
  script:
    - |
      WORKSPACE_NAME=${CI_COMMIT_REF_NAME}
      JOB_ID=${CI_COMMIT_REF_NAME}_job
      sbatch $SCRATCH_BASE/$WORKSPACE_NAME/scripts/submit_workflow_job.sh $SCRATCH_BA
  artifacts:
    paths:
      - $SCRATCH_BASE/$JOB_ID/
```

Listing 5: "Submit Workflow Job" in YAML

2.3.6 Fetch Results

Listing 6 Uses `fetch_results.sh` to retrieve the job results and store them in the repository.

```yaml
fetch_results:
  stage: fetch_results
  needs:
    - submit_ocrd_job
  only:
    - tags
  script:
    - |
      WORKSPACE_NAME=${CI_COMMIT_REF_NAME}
      JOB_ID=${CI_COMMIT_REF_NAME}_job
      bash scripts/fetch_results.sh $SCRATCH_BASE $JOB_ID results/$WORKSPACE_NAME
  artifacts:
    paths:
      - results/
```

Listing 6: "Fetch Results" in YAML

2.3.7 Workspace Cleanup

Implement cleanup procedures to delete or move data from the HPC scratch space after job completion to maintain a tidy working environment.

2.4 GitLab Variables

The following variables need to be set in GitLab Settings for CI/CD Pipeline:

- SSH_PRIVATE_KEY: Your private SSH key content
- SCRATCH_BASE: Path to scratch base
- HPC_USER: HPC username
- HPC_HOST: HPC host
- GITLAB_USER: GitLab username
- GITLAB_TOKEN: GitLab Access Token
- WORKSPACE_NAME: Workspace inside the data directory to be processed.
- WORKFLOW_ID: Workflow file name inside the workflows directory to be used in processing
- C (CPU)_COUNT: Number of CPUs cores
- RAM_SIZE: Number of RAMs Gigabytes

2.5 Nextflow OCR-D Workflow

The OCR-D workflow is parallelized to leverage the computational power of the HPC:

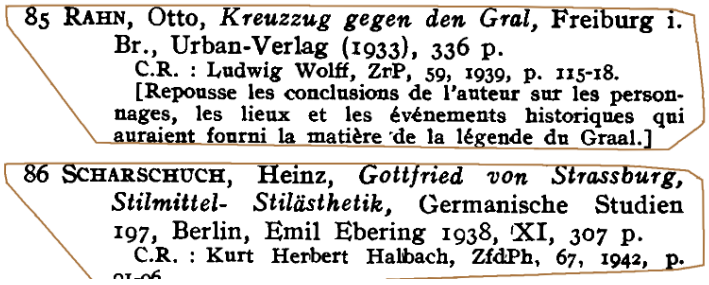
- Data Chunking: The whole amount of pages is broken into smaller chunks, which are handled individually. The chunking process is controlled by dividing the pages into ranges, which allows many processors to handle various ranges at the same time.
- Resource Allocation: Each chunk is allocated a portion of the total CPUs and RAM. For example, if 64 CPUs and 128G (GB) RAM are available and the work is divided into eight pieces, each chunk can utilize 8 CPUs and 16GB RAM.
- Concurrent Processing: All phases of the OCR workflow (e.g., binarization, cropping, denoising, deskewing, segmentation, dewarping, and text recognition) are carried out simultaneously across many chunks. This guarantees that numerous pages are handled concurrently, which considerably reduces total processing time.
- Synchronization: The outputs of each stage are synced. After separate page ranges are analyzed, the data are combined to produce the final output.

3 Results

The OCR-D findings are delivered in Page E (XML) format, which may be difficult to comprehend immediately. To improve readability, the findings are shown using LAREX, an open-source Page XML reader. The OCR-D results show three different stages:

3.1 Segmentation

During this stage, the scanned page is divided into different sections such as text blocks, pictures, and tables. Each area is recognized and defined to assist in distinguishing between different forms of material. Figure 2 depicts the segmentation results, displaying how the page content is divided into distinct sections.


85 RAHN, Otto, *Kreuzzug gegen den Gral*, Freiburg i. Br., Urban-Verlag (1933), 336 p.
C.R. : Ludwig Wolff, ZrP, 59, 1939, p. 115-18.
[Repousse les conclusions de l'auteur sur les personnages, les lieux et les événements historiques qui auraient fourni la matière de la légende du Graal.]

86 SCHARSCHUCH, Heinz, *Gottfried von Strassburg, Stilmittel- Stilästhetik*, Germanische Studien 197, Berlin, Emil Ebering 1938, XI, 307 p.
C.R. : Kurt Herbert Halbach, ZfdPh, 67, 1942, p.

Figure 2: Region Segmentation Results.

3.2 Line Detection

After segmentation, line detection is used to identify individual lines of text inside text blocks. This phase is critical for structuring the text and preparing it for further processing. Figure 3 displays the line detection findings, with highlighted lines within the text blocks.

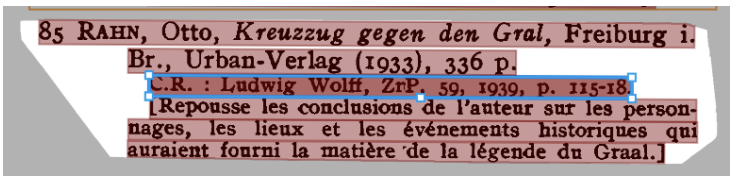


Figure 3: Line Detection Results.

3.3 Text Recognition

The last stage involves converting the identified lines of text into machine-readable text. This entails identifying certain letters and phrases inside each line. Figure 4 displays the text recognition results, which include the recovered text from the scanned paper.

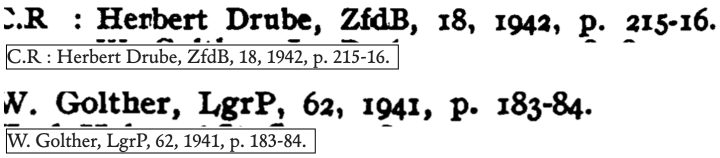


Figure 4: Text Recognition Results.

3.4 The Processing Time

Due to variations in computing power and parallel processing capabilities, the processing time and efficiency of HPC and a standard PC might differ dramatically. Table 1 shows a comparison based on sample data. In Table 1, 128GB of RAM and 64 CPU cores were allocated in the HPC, and 16GB of RAM and 8 CPU cores were allocated in the PC.

Table 1: Comparison of Processing Times on HPC and PC.

Pages Number	HPC Time(min)	PC Time(min)
1	15	6
10	16	9
50	21	25
100	33	64
300	60	181

Figure 5 identifies the relation between the number of pages and the processing time on HPC and PC

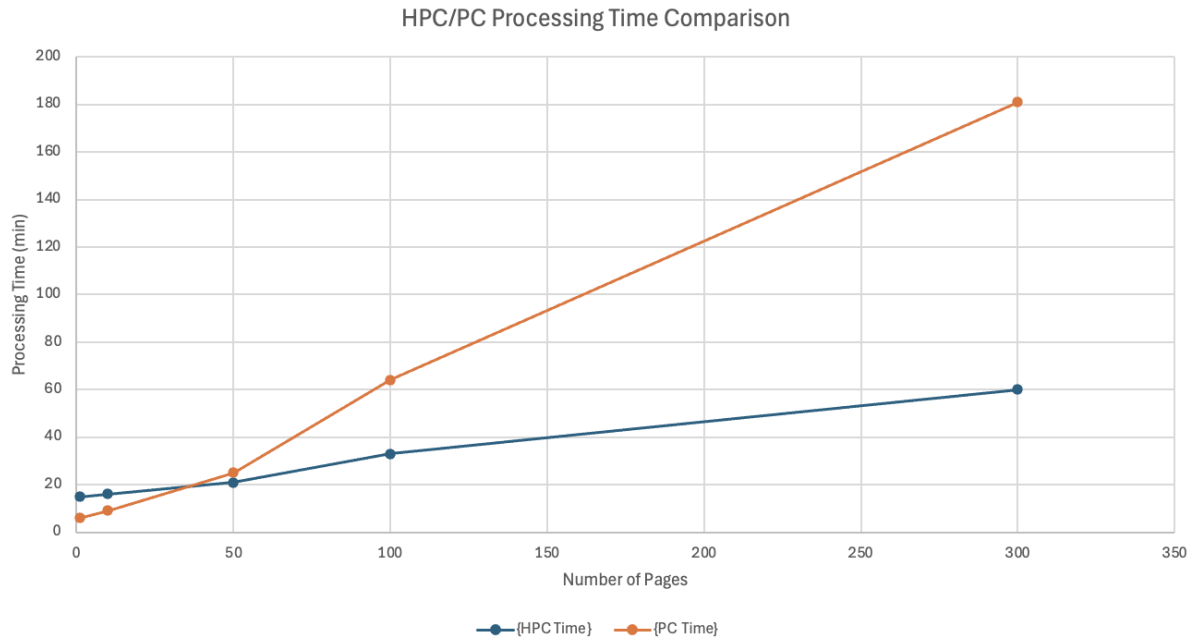


Figure 5: Time Comparison Chart.

4 Discussion

The incorporation of DevOps approaches into High-Performance Computing (HPC) systems, particularly for Optical Character Recognition Development (OCR-D) operations, displays a number of important advances and tangible advantages. This research clearly demonstrates the possibility of using CI/CD pipelines, containerization, and automation to enhance HPC procedures that have previously been manual and error prone.

Efficiency and Speed: The project demonstrates how DevOps methodologies may significantly speed up development and deployment processes. By automating the OCR-D procedures, the system shortens the time necessary to digitize historical documents, which is crucial for projects with huge datasets. The results show that HPC processing times were much faster than conventional PCs, particularly as the number of pages rose. This demonstrates the efficiency benefits that may be achieved by combining HPC resources with DevOps approaches.

Figure 5 illustrates an interesting observation: for a lesser number of pages (less than 35 in the test scenarios), the PC is actually more efficient and quicker than the HPC setup since it requires less time. This phenomenon is likely to be due to the scheduling overhead on HPC systems, which can cause delays that are more visible with lesser workloads. For bigger workloads, however, the HPC's higher processing capacity becomes clear, dramatically lowering total processing time.

Reproducibility and Reliability: Automation using Infrastructure as Code (IaC) makes environments consistent and repeatable. This decreases the possibility of human mistake while increasing the repeatability of outcomes, which is critical in scientific research. The automated pipeline comprises phases for connecting to HPC, downloading required pictures, uploading data, submitting tasks, and receiving results, all of which contribute to a dependable and consistent workflow.

Cooperation and Scalability: The use of centralized version control (GitLab) and automated workflows fosters better cooperation among researchers, developers, and operators. This simplified method enables greater cooperation and shared progress tracking. Furthermore, containerization with Singularity improves scalability, allowing for effective utilization of processing capacity across several HPC resources.

Processing Time Comparison: A comparison of processing times between HPC and regular PC systems demonstrates the huge performance gains given by HPC. For example, processing 300 pages takes 60 minutes on HPC versus 181 minutes on a regular PC. This significant gap highlights the HPC's better capacity to handle large-scale OCR operations effectively.

5 Conclusion

This project successfully incorporates DevOps approaches and HPC settings to optimize OCR-D procedures, resulting in significant improvements in efficiency, repeatability, cooperation, and scalability. Implementing a CI/CD pipeline automates and simplifies the OCR process, greatly lowering processing times and improving result dependability.

The research emphasizes the revolutionary power of merging current software development approaches with high-performance computers. By automating the OCR-D procedure and utilizing HPC resources, the project not only speeds up the digitalization of historical documents, but also provides reliable and repeatable results.

Future work might look at more improvements, such as parallelizing processes for quicker processing, leveraging proxy servers for better speed, and including other tools like LAREX for direct result uploads. This project exemplifies how modern DevOps principles may be successfully applied to scientific computing, laying the groundwork for future advancements in the field.

References

[Cou23] Ludovic Courtès. *Continuous integration and continuous delivery for HPC*. 2023. URL: %5Curl%7Bhttps://hpc.guix.info/blog/2023/03/contiguous-integration-and-continuous-delivery-for-hpc/%7D (visited on 03/06/2023).

[Nuy23] P. Nuyujukian. "Leveraging DevOps for Scientific Computing". In: *Review of Scientific Instruments* (Oct. 2023). URL: https://doi.org/10.48550/arXiv.2310.08247.

[OCR24] OCR-D. *OCR-D Documentation*. 2024. URL: %5Curl%7Bhttps://ocr-d.de/en/%7D (visited on 07/06/2024).

[RMF20] F. Reghenzani, G. Massari, and W. Fornaciari. "Timing Predictability in High-Performance Computing With Probabilistic Real-Time". In: (Nov. 2020). URL: http://dx.doi.org/10.1109/ACCESS.2020.3038559.

[SHH18] Z. R. Sampedro, A. Holt, and T. Hauser. "Continuous Integration and Delivery for HPC: Using Singularity and Jenkins". In: (July 2018). URL: http://dx.doi.org/10.1145/3219104.3219147.

A Code samples

Nexflow Workflow Example

```
nextflow.enable.dsl=2

process split_page_ranges {
    maxForks params.forks
    cpus params.cpus_per_fork
    memory params.ram_per_fork
    debug true

    input:
        val range_multiplier
    output:
        env mets_file_chunk
        env current_range_pages
    script:
    """
    current_range_pages=\$(${params.singularity_wrapper} /
    ocrd workspace -d ${params.workspace_dir} list-page /
    -f comma-separated -D ${params.forks} -C ${range_multiplier})
    echo "Current range is: \$current_range_pages"
    mets_file_chunk=\$(echo ${params.workspace_dir}/mets_${range_multiplier}.xml)
    echo "Mets file chunk path: \$mets_file_chunk"
    \$(${params.singularity_wrapper} cp -p ${params.mets} \$mets_file_chunk)
    """
}

process ocrd_cis_ocropy_binarize {
    maxForks params.forks
    cpus params.cpus_per_fork
    memory params.ram_per_fork
    debug true

    input:
        val mets_file_chunk
        val page_range
        val input_group
        val output_group
    output:
        val mets_file_chunk
        val page_range

    script:
    """
    ${params.singularity_wrapper} ocrd-cis-ocropy-binarize /
 -w ${params.workspace_dir} -m ${mets_file_chunk} --page-id ${page_range} /
```

```
 -I ${input_group} -O ${output_group}
    """
}

process ocrd_anybaseocr_crop {
    maxForks params.forks
    cpus params.cpus_per_fork
    memory params.ram_per_fork
    debug true

    input:
        val mets_file_chunk
        val page_range
        val input_group
        val output_group
    output:
        val mets_file_chunk
        val page_range

    script:
    """
    ${params.singularity_wrapper} ocrd-anybaseocr-crop /
 -w ${params.workspace_dir} -m ${mets_file_chunk} --page-id ${page_range} /
 -I ${input_group} -O ${output_group}
    """
}

process ocrd_skimage_binarize {
    maxForks params.forks
    cpus params.cpus_per_fork
    memory params.ram_per_fork
    debug true

    input:
        val mets_file_chunk
        val page_range
        val input_group
        val output_group
    output:
        val mets_file_chunk
        val page_range

    script:
    """
    ${params.singularity_wrapper} ocrd-skimage-binarize /
 -w ${params.workspace_dir} -m ${mets_file_chunk} --page-id ${page_range} /
 -I ${input_group} -O ${output_group} -p '{"method": "li"}'
    """
```

```
}

process ocrd_skimage_denoise {
    maxForks params.forks
    cpus params.cpus_per_fork
    memory params.ram_per_fork
    debug true

    input:
        val mets_file_chunk
        val page_range
        val input_group
        val output_group
    output:
        val mets_file_chunk
        val page_range

    script:
    """
    ${params.singularity_wrapper} ocrd-skimage-denoise /
 -w ${params.workspace_dir} -m ${mets_file_chunk} --page-id ${page_range} /
 -I ${input_group} -O ${output_group} -p '{"level-of-operation": "page"}'
    """
}

process ocrd_tesserocr_deskew {
    maxForks params.forks
    cpus params.cpus_per_fork
    memory params.ram_per_fork
    debug true

    input:
        val mets_file_chunk
        val page_range
        val input_group
        val output_group
    output:
        val mets_file_chunk
        val page_range

    script:
    """
    ${params.singularity_wrapper} ocrd-tesserocr-deskew /
 -w ${params.workspace_dir} -m ${mets_file_chunk} --page-id ${page_range} /
 -I ${input_group} -O ${output_group} -p '{"operation_level": "page"}'
    """
}

```

```
process ocrd_cis_ocropy_segment {
    maxForks params.forks
    cpus params.cpus_per_fork
    memory params.ram_per_fork
    debug true

    input:
        val mets_file_chunk
        val page_range
        val input_group
        val output_group
    output:
        val mets_file_chunk
        val page_range

    script:
    """
    ${params.singularity_wrapper} ocrd-cis-ocropy-segment /
 -w ${params.workspace_dir} -m ${mets_file_chunk} --page-id ${page_range} /
 -I ${input_group} -O ${output_group} -p '{"level-of-operation": "page"}'
    """
}

process ocrd_cis_ocropy_dewarp {
    maxForks params.forks
    cpus params.cpus_per_fork
    memory params.ram_per_fork
    debug true

    input:
        val mets_file_chunk
        val page_range
        val input_group
        val output_group
    output:
        val mets_file_chunk
        val page_range

    script:
    """
    ${params.singularity_wrapper} ocrd-cis-ocropy-dewarp /
 -w ${params.workspace_dir} -m ${mets_file_chunk} --page-id ${page_range} /
 -I ${input_group} -O ${output_group}
    """
}

process ocrd_calamari_recognize {
    maxForks params.forks
```

```
    cpus params.cpus_per_fork
    memory params.ram_per_fork
    debug true

    input:
        val mets_file_chunk
        val page_range
        val input_group
        val output_group
    output:
        val mets_file_chunk
        val page_range

    script:
    """
    ${params.singularity_wrapper} ocrd-calamari-recognize /
 -w ${params.workspace_dir} -m ${mets_file_chunk} --page-id ${page_range} /
 -I ${input_group} -O ${output_group} -p '{"checkpoint_dir": "qurator-gt4histocr-1.0"
    """
}

process merging_mets {
    // Must be a single instance - modifying the main mets file
    maxForks 1

    input:
        val mets_file_chunk
        val page_range
    script:
    """
    ${params.singularity_wrapper} ocrd workspace -d ${params.workspace_dir} /
    merge --force --no-copy-files ${mets_file_chunk} --page-id ${page_range}

    ${params.singularity_wrapper} rm ${mets_file_chunk}
    """
}

workflow {
    main:
        ch_range_multipliers = Channel.of(0..params.forks.intValue()-1)
        split_page_ranges(ch_range_multipliers)

        ocrd_cis_ocropy_binarize(split_page_ranges.out[0], /
        split_page_ranges.out[1], params.input_file_group, "OCR-D-BIN")

        ocrd_anybaseocr_crop(ocrd_cis_ocropy_binarize.out[0], /
        ocrd_cis_ocropy_binarize.out[1], "OCR-D-BIN", "OCR-D-CROP")

```

```
        ocrd_skimage_binarize(ocrd_anybaseocr_crop.out[0], /
        ocrd_anybaseocr_crop.out[1], "OCR-D-CROP", "OCR-D-BIN2")

        ocrd_skimage_denoise(ocrd_skimage_binarize.out[0], /
        ocrd_skimage_binarize.out[1], "OCR-D-BIN2", "OCR-D-BIN-DENOISE")

        ocrd_tesserocr_deskew(ocrd_skimage_denoise.out[0], /
        ocrd_skimage_denoise.out[1], "OCR-D-BIN-DENOISE", "OCR-D-BIN-DENOISE-DESKEW")

        ocrd_cis_ocropy_segment(ocrd_tesserocr_deskew.out[0], /
        ocrd_tesserocr_deskew.out[1], "OCR-D-BIN-DENOISE-DESKEW", "OCR-D-SEG")

        ocrd_cis_ocropy_dewarp(ocrd_cis_ocropy_segment.out[0], /
        ocrd_cis_ocropy_segment.out[1], "OCR-D-SEG", "OCR-D-SEG-LINE-RESEG-DEWARP")

        ocrd_calamari_recognize(ocrd_cis_ocropy_dewarp.out[0], /
        ocrd_cis_ocropy_dewarp.out[1], "OCR-D-SEG-LINE-RESEG-DEWARP", "OCR-D-OCR")

        merging_mets(ocrd_calamari_recognize.out[0], /
        ocrd_calamari_recognize.out[1])
}
```