SAND2022-3260

SANDIA REPORT
SAND2022-0958
Printed March 9, 2022

Seascape Interface Control Document

Version 3.4

Emily R. Moore
Prolif. Detection Remote Sensing, 6751

Todd A. Pitts
William Marchetto
Henry Qiu
Exploratory Real-Time Sensing, 6773

Leon C. Ross
Prolif. Detection Remote Sensing, 6751

Forest Danford
Space Mission Engineering, 6352

Christopher W. Pitts
Autonomous Sensing & Perception, 5448

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@osti.gov
Online ordering:	http://www.osti.gov/scitech

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.gov
Online order:	https://classic.ntis.gov/help/order-methods

ABSTRACT

This paper serves as the Interface Control Document (ICD) for the Seascape automated test harness developed at Sandia National Laboratories. The primary purposes of the Seascape system are: (1) provide a place for accruing large, curated, labeled data sets useful for developing and evaluating detection and classification algorithms (including, but not limited to, supervised machine learning applications) (2) provide an automated structure for specifying, running and generating reports on algorithm performance. Seascape uses GitLab, Nexus, Solr, and Banana, open source software, together with code written in the Python language, to automatically provision and configure computational nodes, queue up jobs to accomplish algorithms test runs against the stored data sets, gather the results and generate reports which are then stored in the Nexus artifact server.

REVISION HISTORY

Revision	Date	Author(s)	Description
1.0	04-26-21	All authors.	Created
2.0	06-01-21	All authors.	Added clarifying language and new figures
2.1	06-17-21	All authors.	Typos and updated figures
3.0	10-21-21	All authors.	Included updated monorepository
3.1	11-03-21	All authors.	Included multi-stage integration practices
3.2	12-20-21	All authors.	Included full software list to set up Seascape
3.3	01-20-22	All authors.	Included new appendix for development processes
3.3.1	02-01-22	All authors.	Clarified language on pixel versus geo space label identification
3.4	03-09-22	All authors.	Added Docker Containerization instructions

CONTENTS

1. Overview of Seascape 9

1.1. Introduction 9

1.2. A Brief Description 9

1.2.1. Seascape Database 10

1.2.2. Seascape Validation & Verification 11

1.3. System Components and Roles 13

1.3.1. Seascape User Roles 13

1.3.2. System Components 14

2. Using Seascape 21

2.1. Using Seascape by Roles 21

2.1.1. Seascape Maintainer 21

2.1.2. Data Expert 22

2.1.3. Algorithm Developer 23

2.1.4. Program Sponsor 24

2.2. Incorporating a Data Set into Seascape 26

2.2.1. Required Artifacts 26

2.2.2. Ingesting Data into Solr and Nexus 27

2.2.3. Defining a Data Test Set 28

2.2.4. Optional Artifacts 29

2.3. Incorporating an Algorithm into Seascape 30

2.3.1. Prepare a Release of the Algorithm 31

2.3.2. Artifacts and GitLab Structure 32

2.3.3. Creating a Continuous Integration (CI) Schedule for your Algorithm 34

2.3.4. Adding Algorithm Tags 34

References 36

Appendices 37

A. Setting up Seascape From Scratch 37

A.1. Setting up Seascape From Scratch in GitLab 37

A.1.1. Create a user access token 37

A.1.2. Create the Seascape group 38

A.1.3. Create the Seascape-VV subgroup 39

A.1.4. Create a GitLab runner(s) for Seascape 39

A.1.5. Register GitLab runner(s) to Seascape 41

A.2. Putting Your Program in Seascape 44
A.2.1. Create your program space in GitLab 44
A.2.2. Add the Seascape harness to your program 45
A.2.3. Add the Algorithms subgroup 46
A.2.4. Add your algorithms 46

B. Adding Algorithms via Docker **47**
B.1. Seascape Maintainer Responsibilities 47
B.1.1. Prerequisites 47
B.1.2. Gather Information 47
B.1.3. Build and Push Docker Image into Seascape 48
B.1.4. Verify Image in Seascape 48
B.1.5. Runners 48
B.2. Algorithm Developer responsibilities 49
B.2.1. Gather Information 49
B.2.2. Containerize the Algorithm 49
B.2.3. runAlgorithm.sh 49

C. Seascape Core Development Process **51**
C.0.1. Terminology 51
C.0.2. Development Process 52

D. FAQs **54**

E. Software Installation List **56**
E.1. Git 60
E.2. GitLab 60
E.3. GitLab CI/CD 61
E.3.1. Runners 61
E.3.2. Pipelines 61
E.4. Nexus 63
E.4.1. Blob (Binary Large Object) 63
E.4.2. Component 63
E.4.3. Asset 63
E.5. Solr 63
E.5.1. Core 63
E.5.2. Index 64

F. Miscellaneous Notes **65**

G. Example Code **66**

H. Acronyms **67**

LIST OF FIGURES

Figure 1-1. Human-in-the-loop interactions with Seascape 10
Figure 1-2. Basic automated processes within Seascape 12
Figure 1-3. Seascape GitLab group structure 15
Figure 1-4. Test executor automated processes 17
Figure 1-5. Algorithm evaluation automated process 19
Figure 1-6. Report generation automated process 20

Figure 2-1. Final report location in Nexus 24
Figure 2-2. Final report location in Nexus 26
Figure 2-3. Ingest script location in GitLab 28
Figure 2-4. Test definitions location in Nexus 29
Figure 2-5. GitLab group structure for the algorithm-level 32

Figure D-1. Nexus interaction functions within GitLab 54
Figure D-2. Final report location in Nexus 55

Figure E-1. Stage Configuration in .gitlab-ci.yml 61
Figure E-2. Job Configuration in .gitlab-ci.yml 62

Figure G-1. GitLab subgroup location of example files and code. 66

This page intentionally left blank.

1. OVERVIEW OF SEASCAPE

1.1. Introduction

This all-inclusive document describes the components, installation, and usage of the Seascape system. Additionally, this manual outlines the step-by-step processes for setting up your own local instance of Seascape, incorporating new datasets and algorithms into Seascape, and how to use the system itself. A brief overview of Seascape is provided in Section 1.2. System components and the various roles of the intended users of the system are described in Section 1.3. Next, steps on how each role uses Seascape are explained in Section 2.1. Finally, the steps to incorporate data into Seascape-DB and an algorithm into Seascape-VV are outlined in Sections 2.2 and 2.3, respectively. Steps to set up an instance of Seascape can be found in Appendix A.1. Finally, Seascape usage can be found in Section 2.1. The appendix includes code examples, frequently asked questions, terminology, and a list of acronyms.

1.2. A Brief Description

Seascape is an automated test harness that allows an objective, repeatable, and recorded evaluation of machine learning algorithm performance benchmarks. The primary purpose of this system is to produce an unbiased assessment of algorithm performance against real-world data sets. it utilizes standard development practices and git branches to to ensure adequate testing before deploying to production.

The system concept comprises five parts: a well-known, curated, and labeled dataset; a way to provision and configure computer systems for use during evaluations and to define and execute performance tests; a means of assessing the test results; and a framework for recording, reporting, and visualizing the performance assessment. A dataset consists of previously adjudicated data that has been labeled by a subject matter expert. These labels are stored in a simple well-known JavaScript Object Notation (JSON) based text format alongside the data itself. For image data they describe the object or signature type and a polygon enclosing image pixels where the object is present. Algorithms running on other types of non-image data may be evaluated in Seascape as well. Section 1-1 described the basic user interactions with the Seascape system, highlighting its simplicity while noting its majority automated capabilities.

Test definitions (also in JSON format) are used to specify specific elements in datasets. A common use for this is to set up baseline and full test definitions, in order to differentiate between running quick evaluations during a development cycle with the baseline test definition or a more comprehensive evaluation with the full test definition. Test definitions, combined with a tagged algorithm

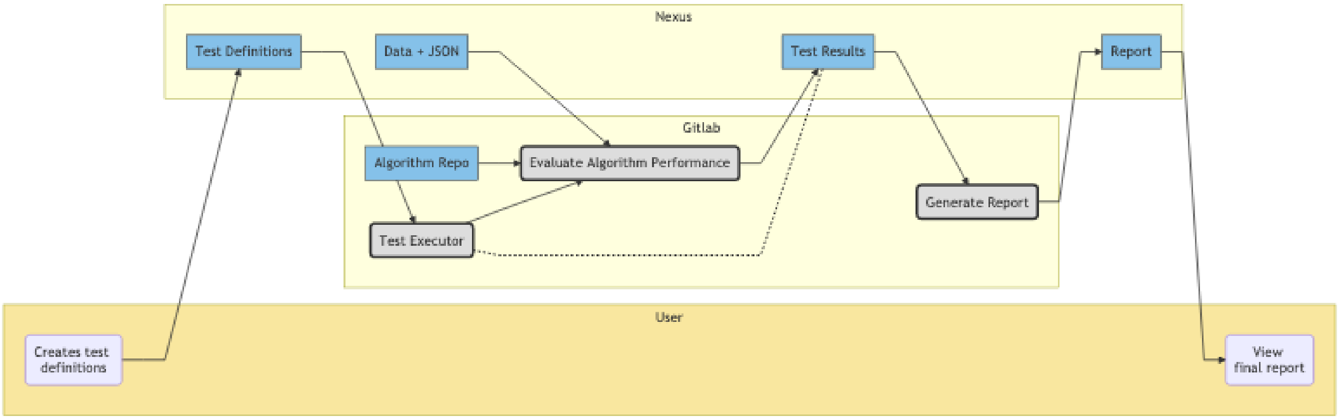

Figure 1-1. The basic user interaction with the Seascape system is highlighted in the dark orange box. After entering a list of data items to define a test in Nexus, the system automatically produces a report for review via a web browser. The light yellow boxes encompass the automated portions of Seascape.

commit in GitLab, define the parameters for a performance pipeline evaluation. The tested algorithm is then run against the specified dataset elements, producing a reported target list. This reported target list is then automatically compared to the data labels per the test definition, yielding a performance benchmark. This benchmark, together with the algorithm version and other details necessary needed to reproduce the test, are saved. Finally, a human-readable report is automatically generated for review. Seascape can be functionally divided into two parts, the database and validation and verification (V&V) infrastructure. These are described in detail below in Sections 1.2.1 and 1.2.2.

Seascape supports the development of both benchmarking and requirements evaluations. Benchmarking evaluation is designed to measure performance of a detection and classification algorithm with a given set of parameters against a specific set of test data. The evaluation contains, for example, false positive rate, recall, precision, or F1 score for a given algorithm against a known target set. This type of evaluation provides the program with expected performance on deployment in a recorded, automated, and repeatable fashion. This is one of the primary purposes of Seascape.

It is possible to define and develop a requirements evaluation within Seascape as well. Requirements evaluations are designed to assess or ensure compatibility and compliance with program and environment requirements. Automated vulnerability or security scans of software or Docker images may be performed in order to ensure compliance with program requirements. An evaluation might also ensure that input and output file formats are compatible with a known standard in preparation for integration into a larger system. Requirements evaluations provide the program with succinct, automated, and repeatable measures of compliance with requirements in support of deployment decisions.

1.2.1. Seascape Database

The database portion of Seascape is known as Seascape-DB. Seascape-DB provides the ability to host and serve data to the Seascape application (or to any other application). It can be queried using a standard web interface. It can also be accessed programmatically and from the command line through a Restful State Transfer (REST) Application Programming Interface (API). Apache Solr, Banana and the Nexus artifact server are open source software applications which provide these capabilities. Seascape-DB can function as intended without Seascape-VV. This is the primary repository for program-collected data and is a critical part of enabling automatic target recognition (ATR)/Machine Learning (ML) in the collected data domains. Details regarding how to format information to provide into the database is covered in Section 2.2. The data set should include all true positive targets as adjudicated by a subject matter expert. The label information must be in a JSON format, with keys defined in Section 2.2. It is the combination of this information which allows the database to function as intended. Once the pair has been ingested into the database, the data can be searched and interrogated via command-line or graphical user interface, and will be made available to all program users specified by an access list. The data access for V&V purposes can be limited as desired. Optionally, supplemental ground truth information or ground truth context may also be included as an additional file. As this data may originate from another source, it might not associate directly with a single piece of raw data. Therefore, it is appropriate for it to reside in its own artifact.

1.2.2. Seascape Validation & Verification

The validation and verification portion of Seascape is known as Seascape-VV. Seascape-VV provides the capability to provision and configure machines, run algorithms in an automated and repeatable fashion, verify the algorithm responses against the labels via Seascape-DB, and report the results. Nexus, GitLab, Python, and a series of additional GitLab and code configuration structures provide this capability. Seascape-VV cannot function as intended without Seascape-DB. The automation capabilities of Seascape-VV create an incredibly easy to use interface for users, as well as provide the capability to repeat and compare algorithm performances against known data.

Provision and configure machines The first critical automation component in Seascape-VV is the ability to provision and configure machines. When an algorithm runs, it will need access to a specific software environment, such as libraries, custom code, and certain versions of software. In addition, algorithms can be written to take advantage of multiple threads or processes to dynamically speed up processing. Seascape-VV offers these capabilities using GitLab runners, discussed in Section 1.3, allowing essentially any runner node to configure a machine to the needs of the algorithm without human intervention.

Define and execute tests Seascape-VV provides an automated capability to define and execute tests. Tests can be defined by any user to generate performance results in an automated fashion for the algorithm. These tests will not only determine the algorithm's response to the data, but will also score the algorithm against the known truth (via the JSON files). The users can define how the response is compared to the labels. For example, an exact pixel-to-pixel match or simply a similar shaped identification.

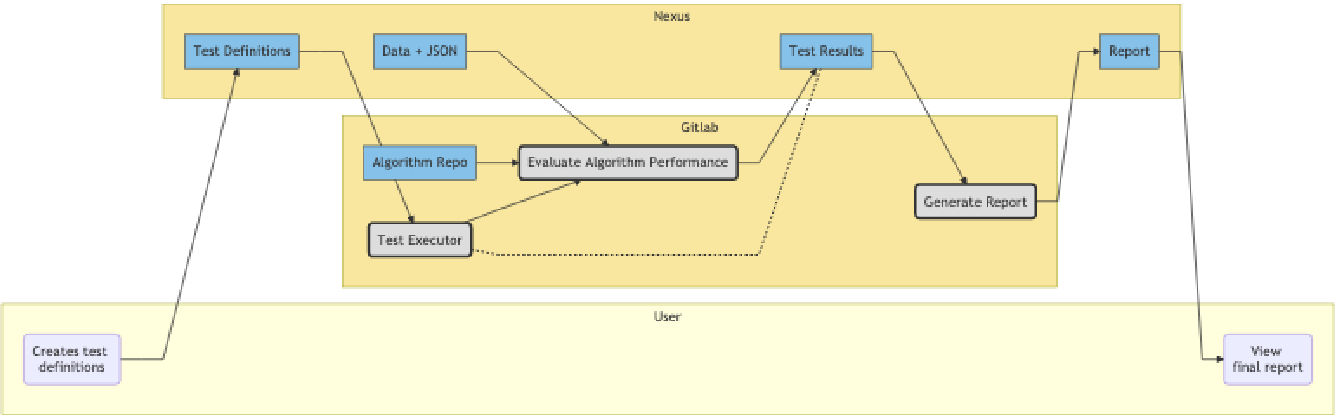

Figure 1-2. The automated sections of Seascape are highlighted in the dark orange boxes. Seascape's infrastructure is primarily automated.

Accessing test results For the current version of any given algorithm, a result is generated, that is, the algorithm's response and the validation against the known truth, for each item in the test data set. Automated processes regularly verify that all data have a result for the current version of the algorithm. When the algorithm changes, new results are generated. These results are available within the Nexus repository, discussed in Section 1.3.

Record, report and visualize performance Performance is automatically assessed through the culmination of results for each data item against the current algorithm version. The automated reporting capability generates a detailed overview and summary results for the entire defined data set. If the report layout needs to change, this can be done without re-running the test results as they are available in Nexus. The report uses numerous result and data visualization techniques to provide a full picture to the users on algorithm performance. The current report breaks down results by target type, and includes the ability to examine image chips in true positive, false positive, false negative, and misclassification categories. The report includes click-able hyperlinks to each aspect of the entire pipeline.

Figure 1-2 describes the basic process of Seascape-VV and how the user interacts with the system. The process begins along the bottom left, in the User group. The user begins the process by creating a test definitions files and pushing the file to Nexus (discussed in Section 2.2.3). The user then simply waits for the process to finish, and views the report once it is generated and published in Nexus (discussed in Section 2.1.4). As designed, Seascape-VV performs its functions automatically, requiring minimal interaction with the user.

Once a user defined CI Schedule executes, the Test Executor process (discussed in Section 1.3.2.3) verified is the results are already present in Nexus. If not, it initiates the Evaluate Algorithm Performance process (discussed in Section 1.3.2.3), pulling data from Nexus and the algorithm information from GitLab. The Evaluate Algorithm Performance process publishes test results for each data in Nexus. The Generate Report process (discussed in Section 1.3.2.3) then creates a report from the information provided in Nexus and publishes a final report. The user is then able to view this final report in Nexus.

1.3. System Components and Roles

This section will provide an in-depth outline of each component and its purpose, as well as the intended usage roles.

1.3.1. Seascape User Roles

Usage of Seascape naturally can be divided into four primary roles, discussed below.

Seascape Maintainer This role is held by the party hosting the Seascape application, either for their own local use or as the maintainer role for a larger program. This role will include tasks such as setting up the system, configuring the GitLab runners, instantiating and ingesting data into Solr, and management of the automated runs. This role should have the ability to set up and make changes to any item within Seascape.

Data Expert This role is held by the users of Seascape for in supporting a curated database as well as assisting automated V&V. This role will include tasks such as preparing and ingesting the data into the database, working with other roles to provide expert recommendations as to which data should be used for V&V testing, how the data should be validated by the Seascape-VV CI pipelines, and continued ingestion. This role should not have the ability to alter anything within the algorithm areas, and should only work within the community code repositories for data management tools.

Algorithm Developer This role is held by the users of Seascape for the purpose of automated algorithm V&V. This role will include tasks such as providing the data and data schema information to the maintainers, providing their algorithm and required artifacts to the maintainer, and reviewing the final reports. This role should have the ability to make changes only within their algorithm area and should not have the ability to alter any other pieces of Seascape.

Program Sponsor This role is held by the sponsor of Seascape for the purpose of running automated validation and verification pipelines regularly. This role will include tasks such as defining data test sets for runs, approving the final report structure, and running tests manually or automatically. This role should involve regular interactions with the maintainers, and should allow the maintainers to make changes on their behalf. This role should not have the ability to alter any algorithm area information, while maintaining the ability to view anything within Seascape.

Roles are not strictly enforced, though, access to various levels of the GitLab repository may be restricted based on roles. As certain roles will have inherent access to all data within the system, please work with the Seascape maintainer and/or the Program sponsor to put in place an non-disclosure agreement (NDA) if desired or appropriate.

1.3.2. System Components

Seascape is comprised of three categories of components which work together to provide its unique capabilities:

- Software stack
- GitLab group structure
- Configuration code repositories

1.3.2.1. Software stack

The following modern software programs were chosen for their merits in interoperability, availability, open nature, and security considerations.

Git Git is the version control system of record for Seascape. Teams and individuals are free to use whatever tooling they want to develop their algorithms, but the evaluations against the official test data sets are performed on code checked into the Git repositories. Git provides the basis for automated pipeline testing, as well as version control and documentation which allows for a better record of testing and repeatability of tests. Common terminology can be found in Appendix E.1.

GitLab GitLab is an integrated version control, testing, and deployment suite of tools, built on top of the Git version control system. The current version of Seascape is built in GitLab, leveraging many of the built-in tools. GitLab provides a practical user interface to perform Seascape tasks, observe the pipeline during runtime, and view subsequent artifacts in a more user friendly manner. Common GitLab terminology used throughout this document, such as commits and tags, can be found in Appendix E.2.

GitLab CI GitLab CI is a tool for Continuous Integration and Delivery (CI/Continuous Development/Deployment (CD))[4]. The general idea behind continuous integration and delivery is that code should be tested often and quickly. This can help assure that newly generated code will not modify or corrupt previously correct processes. When using a version control system like Git, CI/CD generally runs tests on each commit, when it is pushed to the repository. Common terminology can be found in Appendix E.3.

```
Seascape
└─Seascape-VV
    ├─<Your Program>
    │  └─Algorithms
    │     └─<Your Algorithm>
    ├─Seascape-Core
    └─ExampleAlgorithms
```

Figure 1-3. The basic Seascape GitLab group structure. The top level is referred to as the Seascape-level, with the subsequent level called Program-level, and then the Algorithm-level. Each user role has an assigned area and permissions into levels are based on those roles.

GitLab Runner A GitLab runner is an application used by GitLab CI/CD to run jobs in a pipeline. Runners are used throughout Seascape, for example, to provision and configure machines to evaluate algorithm performance. GitLab runners are registered at the Seascape level. Each runner will commonly used research & development (RD) packages, as well as a Linux operating system (OS), the specifics of which are dependent on the environment where Seascape is being hosted. Additional, algorithm specific requirements can be added to all or specific runners, with descriptive tags indicating which runner has any given specific item, such as Graphics Processing Units (GPUs) or a different OS.

Nexus Nexus is an artifact repository manager. An artifact repository provides a seamless way to collect and share artifacts, while abstracting away the artifact's physical location on disc. Nexus serves artifacts from Seascape-DB, as well as test definitions, test results, and reports generated from Seascape-VV. Common terminology can be found in Appendix E.4

Solr/Banana Solr is a metadata database used by Seascape-DB's data search and discovery. Solr provides a distributed search and indexing functionality of Seascape-DB data. It easily serves the data into Seascape-VV in an automated and programmatic fashion. Solr provides the backend information to Banana, a Graphical User Interface (GUI) to display faceted information that is easily customizable to the users preference. Banana provides a quick visual interrogation tool to understand the data within Seascape-DB.

Conda/Python The configuration code repositories which make up Seascape are written in the Python 3 language. Seascape uses Anaconda python to manage and install the Python packages necessary for Seascape.

1.3.2.2. GitLab group structure

Figure 1-3 described the required GitLab group structure for a general program with one algorithm. The program level contains a variety of algorithm repositories. Each algorithm repository is self-contained for testing and reporting.

Seascape-VV level The Seascape structure level houses most commonly used code and capabilities needed to create and manage Seascape. Only those with the maintainer role should have the ability to change any information at this level. Data experts will need access to this level, though should not change any information at this level.

Program level The program level consists of program named folder, and houses all information pertinent for integration with the Seascape system. The Algorithms area is defined below. While it is appropriate for both the maintainer and program roles to have change permissions at this level, the maintainer role should make all necessary changes.

Algorithm level The algorithm level consists of all algorithms within a given program. Each algorithm will be provided with its own named folder and will include the algorithm repository and required artifact codes provided by the algorithm developer role. More information can be found in Section 2.3. The algorithm developer role should be the primary user at this level.

1.3.2.3. Seascape-Core

Additional code that is necessary to utilize the full capability of Seascape can be found in Seascape-Core. This repository consist of scripts to enable items such automated installation of software and configuration files, enable automated testing, and other important items. The Seascape-Core repository should be placed under the Seascape-VV GitLab structure, as shown in Figure 1-3.

Test Executor The Test Executor primarily includes python code coupled with a GitLab CI/CD file to create the primary automation functionality of Seascape-VV. Figure 1-4 outlined the decision tree of the Test Executor. The Test Executor will run on a user defined schedule. Details of this set-up are discussed in Section A.

Figure 1-4 should be interpreted starting from top with the Wake Up node. The process begins by pulling test definitions from Nexus. If there are no test definitions, the process shuts down, as there will be no report to generate. If a test definitions file is present, next the process looks for an algorithm commit tags from GitLab. Specifically, the process is looking for a commit tag within the algorithm repository to indicate which algorithm version and test definitions file was used. If there is no algorithm commit tag present, the process shuts down. If the tag is present, the process will then look for test results in Nexus for that algorithm commit tag. If there are no test results, the process will create the pipeline necessary to generate those results. If there are test results present, the process will verifies there were no failures present, through the presence or absence of a `failure.txt` file within Nexus. If there was a failure present, the process shuts down. If there was not, the process checks to see if test results are complete, that is, there are test results for each piece of data being evaluated. If they are, then this process has completed its tasking and shuts down. If not, the process will create the pipeline necessary to generate the results. This pipeline will include starting the process to evaluate the algorithm, discussed in Section 1.3.2.3 and in Figure 1-5.

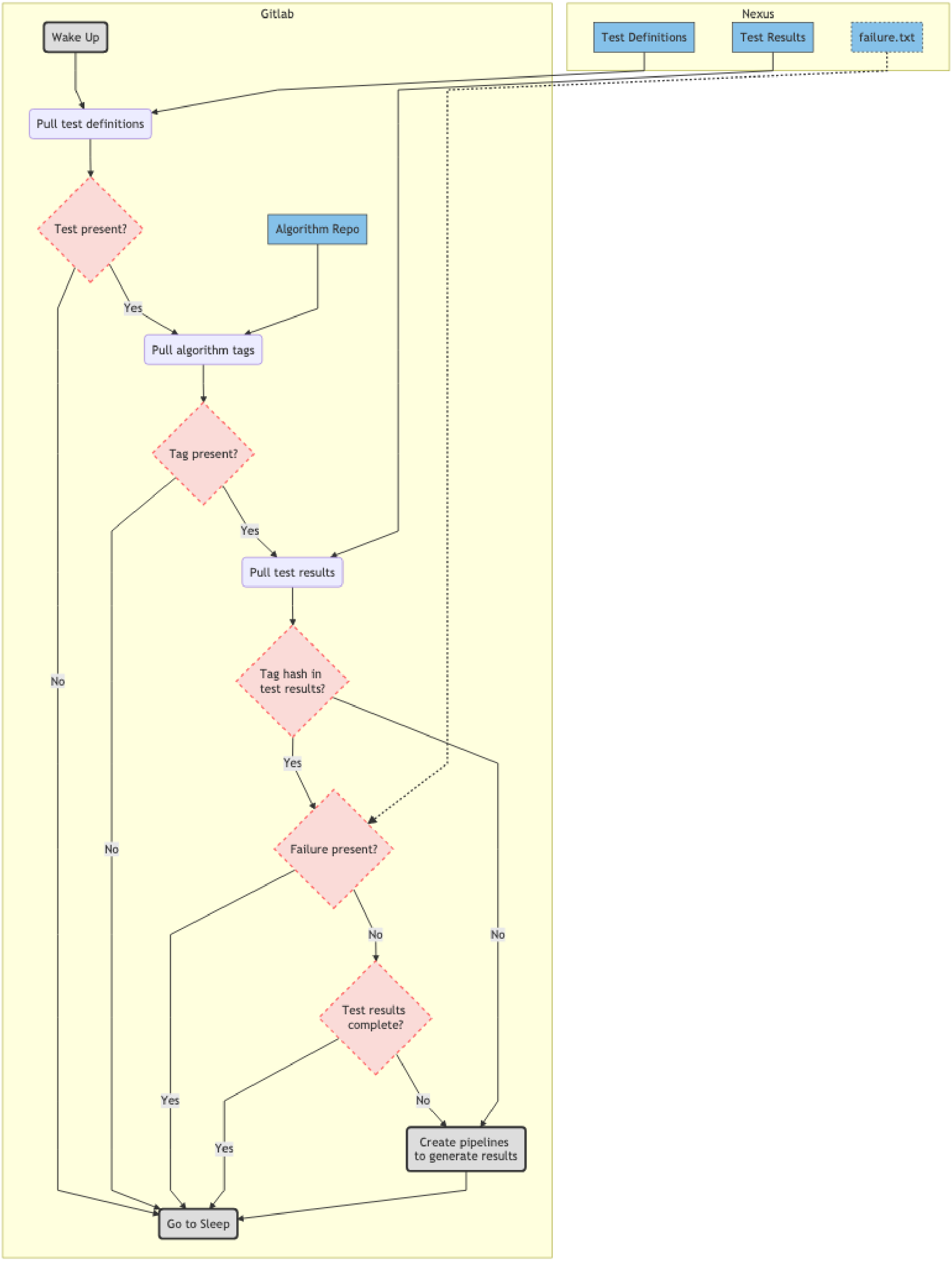

Figure 1-4. The basic functions of the automated test executor can be broken down into a decision tree. Test executor is the heartbeat of the automated Seascape processes.

A major benefit to the design of this process is that it will automatically comb through Nexus and make sure every results has already been generated for the information requested - regardless of whether that information was requested a minute ago, a day ago, or a year ago. This means, as the database of relevant data within Seascape-DB grows, this process will automatically generate pipelines to evaluate the algorithm on the new test data. This also allows for the more lengthy processes, such as algorithm evaluation, to run on a regular schedule, to include nights and weekends. Finally, when a new algorithm version has been committed and tagged appropriately (as opposed to simply committing), this process will ensure all pipelines to generate results are kicked off, without the user having to do so. Allowing the flexibility to include the tag or not means results will not get regenerated for small, untagged commits, like a new comment, or print statement.

Performance Performance consists python scripts needed to generate performance reports. The capabilities of performance are cued from the test executor process. Figure 1-5 describes the process of evaluating algorithm performance for Seascape-VV.

Figure 1-5 should be interpreted starting from top, at the Test Executor trigger node. This process is kicked off by the test executor process, described in Section 1.3.2.3. The test executor triggers the performance pipeline CI for the algorithm and provides the algorithm commit hash and data IDs to this process. Any specific GitLab runner requirements listed within the performance pipeline CI are met at this step. For example, if the algorithm needs a Windows OS with Central Processing Unit (CPU) clusters available, the performance pipeline CI will list both the OS and hardware as requirements. Then, Seascape-VV will select a specific GitLab runner tagged with these requirements and run the process on that specific runner and associated runner node. The algorithm is then cloned and checked out into that machine using the algorithm information from the algorithm repository in GitLab. Next, the data is pulled from Nexus and paths are generated to the data in the `IFILE` environment variable, discussed in detail in Sections 2.2.3 and 2.3. A shell script, defined for each algorithm, discussed in Section 2.3, reads in the `IFILE` environmental variable and executes the algorithm against the data listed within `IFILE`. Algorithm results are generated and set within the `OFILE` environmental variable, discussed in Section 2.3, and the shell script ends. Results are then checked using the known truth from Nexus against the results generated for each piece of data and are collected in a single file, `results.json`, published in Nexus. At any time, if a node fails, to include the algorithm, a `failure.txt` file is generated and published in Nexus, which includes information as to the nature of the failure.

A major benefit in the design of this process is that it remains independent of the final report process. This means that results do not need to be regenerated if there is a format change, data ID list change, or otherwise small change made to the report or pipeline. This keeps potentially long processes from regenerating results without the technical need to.

Reports Reports consists of python scripts needed to generate performance reports. The capabilities of the report generator process is initiated by same defined GitLab CI/CD scheduler as Test Executor. Details of this set-up are discussed in Section A. Figure 1-6 describes the automated report generation process for Seascape-VV.

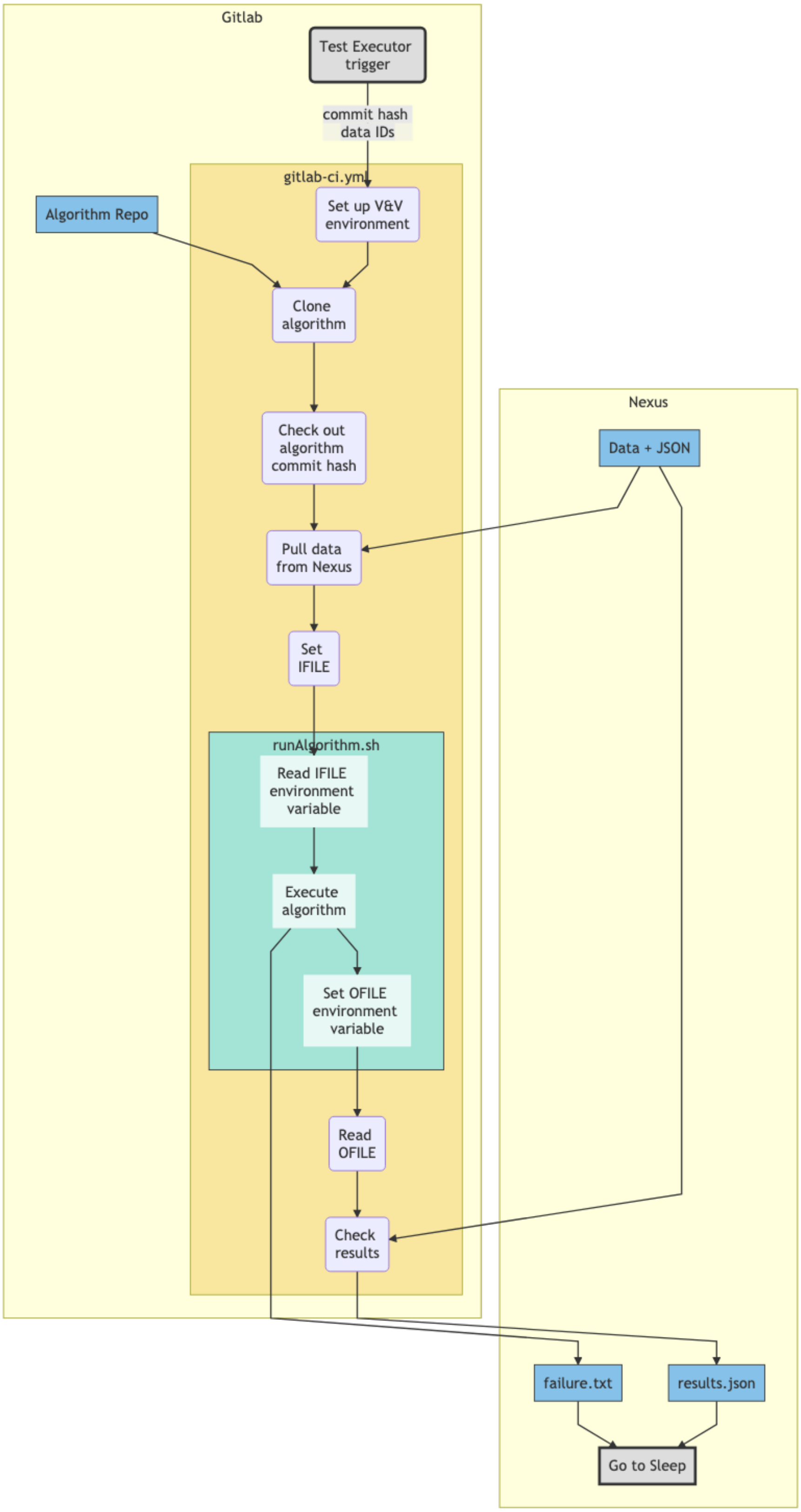

Figure 1-5. The basic functions of the automated algorithm evaluation processes can be broken down into a decision tree. These processes are at the center of Seascape, running and evaluating algorithm performance.

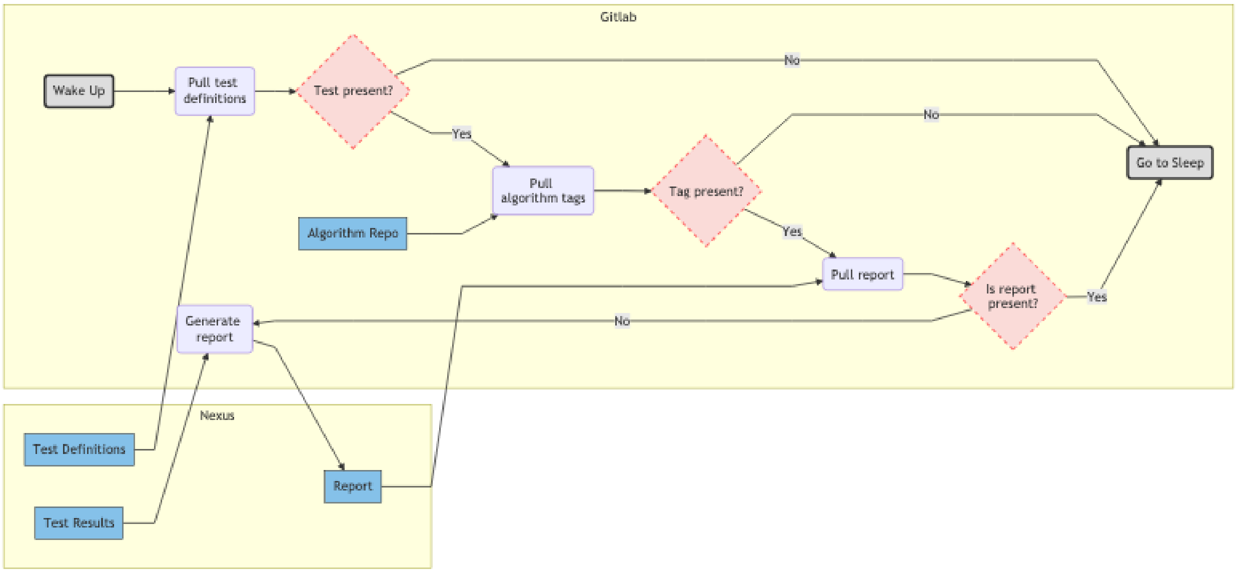

Figure 1-6. The basic functions of the automated report generation processes can be broken down into a decision tree. This processes is is the final automated step of the system, generating new reports based on previously gathered results.

Figure 1-6 should be interpreted starting from left hand side with the Wake Up node. The process begins by pulling test definitions from Nexus. If there are no test definitions, the process shuts down, as there will be no report to generate. If a test definitions file is present, next the process looks for an algorithm commit tags from GitLab. Specifically, the process is looking for a commit tag within the algorithm repository to indicate which test definitions file was used. If there is no algorithm commit tag present, the processes shuts down. If the tag is present, the process will look in Nexus for a report from that specific tag. If there is a report already present, the processes shuts down. If there is no report present, the process will generate a report using the test results from the algorithm commit tag in Nexus. The report is then pushed to Nexus and the process is complete.While this process can be initiated manually or on a schedule, it naturally is subsequent to the test executor and execute algorithm performance processes.

A major benefit to the design of this process, is that if there are test results for a specific algorithm commit tag already, but a user would like to see a report from a certain subset of those results, the test results do not need to be regenerated in order to create the report. Simply put, the report generation process cumulates already calculated results into one file. This saves processing time if the user decides to remove or add data which already has results in Nexus.

Ansible This repository contains the Ansible script to install instances of Nexus, Solr, and GitLab. Installation can be for each item independently or a combination based on user's needs.

Common This repository contains commonly shared code used by other Seascape repositories and pipelines. For example, the ingest and interface code for Solr and Nexus resides in this repository. This repo also contains the Python API for gitlab.

2. USING SEASCAPE

2.1. Using Seascape by Roles

The skills and work flow to use Seascape are highly dependent on which role you have in the process. In this section, we discuss the prerequisite skills and common work flow tasks as defined by the roles identified in Section 1.3.1.

2.1.1. Seascape Maintainer

Work flow:

- Install and configure Seascape
- Prepare the program level in GitLab
- Add users to GitLab program group
- Provide ICD to users
- Ingest new data into Seascape-DB
- Set requirements for algorithms
- Create a new CI schedule for each algorithm

The following section describes the primary work flow of the Seascape maintainer and where to find detailed information on the tasks. In additional to the tasks below, the Seascape maintainer is expected to be the expert on the system and provide assistance to all users, while maintaining the infrastructure.

Install and configure Seascape The first step is to install and configure Seascape. Details and prerequisites can be found in Appendix A.

Prepare the program level Next, create a space within Seascape specific to the program. Details and prerequisites can be found in Appendix A.2.

Add users Users need to be added to the relevant program group. This will provide them access to interact and use the system. The users must already have an account in GitLab to perform this step.

Provide ICD Provide the ICD, located in the Documents subgroup, shown in Figure 1-3 in Section 1.3.2.2, to all new users.

Ingest new data into Seascape-DB Once the data expert has provided the data, with labels, and a Solr schema file, the data is ready to create a Solr core, and therefore viewable by Banana, as well as get ingested into Nexus. Details can be found in Section 2.2.

Set requirements for algorithms Some algorithms may require a specific OS, hardware, software, or other requirements to run. These requirements should be levied through which machines are available for GitLab runners to use. The Seascape maintainer is responsible for making those requirements available via registered GitLab runners, working with the hosted environment. If any requirements are not available, the algorithm developer needs notified immediately. If algorithms require docker, please consult Appendix B.1.

Create a new CI schedule for each Algorithm In order to add an algorithm to the automated capabilities of GitLab CICD, any new schedule must be added to Seascape-Core pointing to the specific algorithm. This is a task to be performed by the maintainer role. Details can be found in Section 2.3.3.

2.1.2. Data Expert

The following section describes the primary skills required and typical work flow of the data expert and where to find detailed information on the tasks. Please take a moment to read the ICD, located in the Documents subgroup, shown in Figure 1-3 in Section 1.3.2.2.

Prerequisite skills:

- Generate an Extensible Markup Language (XML) file to define the Solr schema
- Create a script to fuse data into a single Solr document

Work flow:

- Get added to GitLab group
- Provide data to maintaner for ingest
- Provide data assistance

Get added to GitLab group In order to use and interact with the Seascape system, your GitLab user must be added to the relevant program group within GitLab. Please provide this information to the Seascape maintainer.

Provide data for ingest Data must be prepared and curated for ingest into Seascape-DB. Details can be found in Section 2.2. Any optional artifacts should be added as well. Anytime new data has been captured and curated, the new information must be added to the Solr core. Examples can be found in Section 2.2. If the data is unique, please work with the Seascape maintainer to encompass the uniqueness of the data.

Provide data assistance As the data expert, this role is expected to provide support to both the algorithm developer and the program sponsor when creating test definitions, determining machine learning test sets, and other data related tasks.

2.1.3. Algorithm Developer

The following section describes the primary skills required and typical work flow of the algorithm developer and where to find detailed information on the tasks. Please take a moment to read the ICD, located in the Documents subgroup, shown in Figure 1-3 in Section 1.3.2.2.

Prerequisite skills:

- Check in code to GitLab to create and update the algorithm repository
- Generate a shell script wrapper to execute the algorithm against data

Work flow:

- Get added to GitLab group
- Provide requirements for algorithm
- Prepare test definitions files, if desired
- Add algorithm and artifacts into appropriate algorithm folder
- Commit any new algorithm changes, adding a tag when necessary
- View the report

Get added to GitLab group In order to use and interact with the Seascape system, your GitLab user must be added to the relevant program group within GitLab. Please provide this information to the Seascape maintainer.

Provide requirements for algorithm Some algorithms may require a specific OS, hardware, software, or other requirements to run. These requirements should be levied through which machines are available for GitLab runners to use. Algorithm developers should provide a list of requirements to the program sponsor. These requirements are then communicated to the Seascape maintainer, who will ensure requirements are made available via registered GitLab runners.

```
  Reports
└─<Your Program>
   └─Algorithms
      └─<Your Algorithm Name>
         └─<Your Algorithm Hash>
            └─<Your Test Definitions Name>
               └─report.html
```

Figure 2-1. Location of the final report within the Nexus artifact structure.

Prepare test definitions Test definition are used to specify a subset of samples included in a single evaluation. As an algorithm developers, it might be desirable to create your own test definitions set for rapid integration or using a partial partition. If you are unfamiliar with the data, please work with the data expert on this task. Creating and pushing test definitions can be found in Section 2.2.3.

Adding an algorithm and artifacts Next, the algorithm and artifacts must be added. As discussed in Section 2.3, a new GitLab project must be created for each algorithm. Then, the algorithm repository including the Seascape compliant algorithm, as well as required artifacts, can be added. Once this step as been completed, please inform the Seascape maintainer, as they will need to add the new algorithm into test executor in order for the automated processes to include the algorithm. If your algorithm requires docker, please refer to Appendix B.2.3.

Commit algorithm changes Algorithm changes can be committed and synced to the GitLab origin at the developer's discretion. In order to trigger an evaluation pipeline, simply tag a commit using the naming convention detailed in Section 2.3.4. any time a commit is tagged, it is recommended that the algorithm developer validates the results.

View report Whenever generated, via creating a test definition or tagging a commit, evaluation reports should be viewed to validate results. The report is found in Nexus in the following location:

2.1.4. Program Sponsor

The following section describes the primary skills required and typical work flow of the program sponsor and where to find detailed information on the tasks. Please take a moment to read the ICD, located in the Documents subgroup, shown in Figure 1-3 in Section 1.3.2.2.

Prerequisite skills:

- Use a browser to navigate the Nexus repository to locate test definitions and final report artifacts

- Delete and/or push a simple text file into Nexus using the browser to generate new evaluation runs
- Fill out a simple text template to define the test definitions

Work Flow:

- Get added to GitLab group
- Define test definitions for V&V testing, with data expert
- Communicate requirements with algorithm developer and Seascape maintainer
- Define automated processes run schedules, with Seascape maintainer
- View the reports

Get added to GitLab group In order to use and interact with the Seascape system, your GitLab user must be added to the relevant program group within GitLab. Please provide this information to the Seascape maintainer.

Define test definitions Often, the program sponsor will chose a select subset of data for each algorithm to perform for V&V testing. While evaluating an algorithm against the entire dataset does not require a test definitions file, running on a subset does. This subset of data should be chosen with the data expert, as well as experts in algorithm V&V. The program sponsor should fill out the test definitions example template file found in Appendix G. The file will then need to be pushed to Nexus. More details can be found in Section 2.2.3.

Communicate requirements Some algorithms may require a specific OS, hardware, software, or other requirements to run. These requirements should be levied through which machines are available for GitLab runners to use. Program sponsors should work with the algorithm developers to determine if and what requirements are necessary. GitLab runners are registered at the Seascape level, and therefore should be registered, added, and maintained through the Seascape maintainer. The program sponsor should communicate these requirements to the Seascape maintainer to implement.

Define automated process schedules Two of the three automated processes within Seascape run on a predetermined schedule. These can be altered to meet the needs of the program sponsor. Please work with the Seascape maintainer to make the appropriate selection and codify the schedule in GitLab.

View report The evaluation report should be viewed regularly to validate results. The report is found in Nexus in the following location:

```
  Reports
└─<Your Program>
   └─Algorithms
      └─<Your Algorithm Name>
         └─<Your Algorithm Hash>
            └─<Your Test Definitions Name>
               └─report.html
```

Figure 2-2. Location of the final report within the Nexus artifact structure.

2.2. Incorporating a Data Set into Seascape

In order for Seascape to perform V&V against an algorithm, a properly curated data set must be provided for reference. This section is primarily for the data expert role. The data set is accessed programmatically by Seascape using Nexus and made available for command line or GUI using Solr and Banana. A Solr core is created for each data set, given the appropriate artifacts. Once the artifacts have been created, the data can be pushed into Solr and Nexus.

2.2.1. Required Artifacts

The following artifacts are required from the data expert in order to prepare for ingest into Solr and Nexus:

- Raw image data
- Label data in JSON
- Solr schema definition in XML

Raw data must be provided in its raw format, with an accompanying label data JSON file with an item name. For example, images with a raw Tag Image File Format (TIFF) format could be named *Image20200620-00954Z.tif* with an associated label file *Image20200620-00954Z.json*. The label data JSON will include a list of all labels within a single datafile. Any other additional information is permitted in the JSON file. Each label in the label data JSON will include the following required keys[1] :

```
featureID: <string>
   Note: Unique value across the entire database, for example UUID

class: <string>
    Note: Class definition, can be as generic as "target"
```

[1] While the data expert may know that there is absolutely no ambiguity in arriving at lat-lon coordinated in relation to the lat-lon present in the label (polygonGeo) and could therefore score in lat-lon space, there can be ambiguity present in other algorithms when they arrive at lat-lon space (e.g. choosing an affine vs. perspective transform to go from pixel space to lat long space). Therefore, some programs have required scoring in the native pixel space to eliminate any ambiguity to allow for appropriate comparison in algorithm performance for the same dataset. Please check with your program.

```
imageID: <string>
   Note: Associated image or data file name, without the full path

polygonPxl: <list of float pairs>
   Note: Polygon corners in (x,y), also known as (column, row),
   pixel location. This should be a closed polygon (first/last point
   will be the same)
```

If the raw data is geo-located, then each label in the label data JSON will also include the following keys:

```
polygonGeo: <list of float pairs>
   Note: polygon corners in (longitude, latitude) decimal degree
   location.  This should be a closed polygon (first/last point will
   be the same) and the corners should be in the same order as
   the polygonPxl list.

geoTransform: <list of float triplets>
   Note: the 3x3 transformation matrix used to convert polygonPxl
   pixel coordinates to polygonGeo geo-coordinates.
```

An example of the label JSON data can be found in Appendix G. Once the raw data and label pairs have been generated, the data needs to be prepared to generate a Solr core. Solr requires a schema file to describe the database format. The schema file is unique to each data set and therefore must be generated by the data providers. The Solr schema file is an XML file which describes the data that Solr will index. The schema defines a collection of fields, both the field name itself and field type are specified. Field type definitions are powerful and include information about how Solr processes incoming field values and query values. An example of the Solr schema file can be found in Appendix G. Generally, this function should be performed by the data expert.

2.2.2. Ingesting Data into Solr and Nexus

Once the required artifacts have been prepared, the following steps are performed by the Seascape maintainer role to ingest the data into Solr and Nexus:

- Fuse data into a single Solr database document
- Post database document to Solr
- Ingest data into Nexus

Generally, this function is performed by the maintainer, who is responsible for Solr and Nexus. First, data must be fused into a single Solr database document. This is a critical step in identifying what information needs to be provided to Solr for the given data set, given future queries. In the previous section, we identified the label JSON as important and required information. There is likely additional information the developer or program would like to query on, such as where the image is located on the planet or what time the image was taken. This is the step in which we define additional information for future queries.

```
Seascape
└─Seascape-VV
  ├─<Your Program>
  │ └─Algorithms
  └─Seascape-Core
    └─Common
      └─Ingest
        ├─ingest_dataset.py
        └─ingest_nexus.py
```

Figure 2-3. GitLab subgroup location of the generic ingest scripts.

If optional artifacts are available, such as those described in Sections 2.2.4.1 and 2.2.4.2, this additional information can be fused during this step as well. The artifact which enables this step is a list of dictionaries to serve as the single database document for Solr. Next, the database document must be posted to Solr. Finally, the data must be ingested into Nexus. These are easy, common commands that can be performed in a few ways. Generic ingest scripts for both Solr and Nexus can be found in Figure 2-3.

If these scripts do not meet the data or optional data specific needs, you must create your own script. If you do create your own script, work with the Seascape maintainer to add your script into the repository. An example of a more specific ingest script is available in Appendix G, showing, in Python, how to create the database document for Solr, post the document to Solr, and ingest the data into Nexus. We include examples of adding optional information, such as the image time from the image and sensor geometry from the image metadata.

2.2.3. Defining a Data Test Set

In this section, we discuss how to define which data an algorithm will be evaluated against. There are two primary sets of data against which algorithms are run: the entire or full dataset and a, generally much smaller, subset of data. The smaller subset is also called the baseline set. A baseline is commonly used to do a quick comparison of results, or provide a quick assessment for the algorithm team. There are two steps to define a test set:

- Create a test definitions JSON file
- Upload to Nexus

As discussed in Section 1.3.2.3, the test executor automated process looks at the algorithm commit tag to determine if a full or baseline subset dataset will be used to evaluate the algorithm. The process then looks at the test definition files in Nexus to determine which dataset name the algorithm tag is associated with. A test definition file also describes which target classes to use in the dataset (some datasets include many target types). The specific images comprising the baseline dataset are also specified in the test definition file. Seascape-VV defaults to the baseline or smaller dataset, preventing users from accidently beginning a lengthy evaluation process against all available data. In order to evaluate against the entire dataset the tag in gitlab should include the five characters

```
  TestDefinitions
└─<Your Program>
   └─Algorithms
      └─<Your Algorithm Name>
         └─<YourTestDefinitions.json>
```

Figure 2-4. Location of the test definitions files within the Nexus tree structure.

_full at the end of the tag name string. If the tag name string ends in *_baseline*, then a smaller, baseline dataset will be used for evaluation.

Create a test definitions JSON file This file is created by hand in any editor the user is comfortable with. An example template file is available in Appendix G for the program sponsor role to complete. The test definitions file provides the automated processes information as to which dataset and which targets within the datasets to use. If a baseline, or subset dataset is desired, those specific baseline data IDs are listed in this file. The test definitions JSON file should have the following keys:

```
dataset: <string>
   Note: Name of the dataset, consistent with the Nexus repository name

target_list: <string>
   Note: List of target types to be evaluated by the algorithm

baselineIDs: list of <strings>
   Note: A list of image names with file extension. This is the
   list of images for baseline or default evaluations. Use of this key
   by the automated system is controlled by the name of the algorithm tag
   in GitLab.
```

An example test definition can be found in Appendix G.

Upload to Nexus This file must be placed within the TestDefinitions folder in the Nexus repository, under the appropriate program and algorithm, as visualized in Figure 2-4.

Numerous baseline test definitions can be generated, each defined by their their own JSON file. If there are numerous baseline test definition files, each one will be evaluated when running in baseline mode.

2.2.4. Optional Artifacts

Additional files are permitted and encouraged to further describe the data, which can be stored in their own Solr core. These files are ensure the all possible pertinent information is found in the database.

2.2.4.1. Additional data metadata

An accompanying metadata JSON file may be provided to further describe the original data. Addition information example include environmental conditions, sensor configurations, technical specifications about the instrument or data, sensor geometry, wind conditions, daylight or nighttime collections or sensor configurations. This filename must match the associated data file name with an *_metadata* marker at the end of the filename. For example, data named *Image20200620-00954Z.tif* would have an metadata file with the name*Image20200620-00954Z_metadata.json*. This JSON file can include as many or as little keys as desired. Additionally, this additional metadata file will be ingested into the Solr data core and into Nexus along with the primary data and label data.

An example metadata JSON can be found in Appendix G. As this information is directly related to a specific data product, it will remain with the data product within the Solr core.

2.2.4.2. Supplemental ground truth data

A supplemental ground truth Hierarchical Data Format 5 (HDF5) or ground truth context file may also accompany the image and label. Supplemental ground truth information is defined as other information pertaining to the location of a target as reported by another entity. For example, your project may include images and labels on the location of a car within that image. The car may have its own position information as reported by a GPS unit. This would be considered ground truth information. This information is often critical for developers during algorithm RD and therefore should be stored alongside the image and label in Nexus.

```
time: <string>
   Note: time in epoch

lat: <float>
   Note: latitude of the target in decimal degrees, from (-90,90)

lon: <float>
   Note: longitude of the target in decimal degrees, from (-180,180)
```

Optional information about the target can be included, if relevant. HDF5 attributes are highly encouraged to properly describe units of measurement. An example ground truth HDF5 can be found in Appendix G. As this information is not necessarily related directly to a specific data product, as not all ground truth information has an associated data product, it will get stored as its own separate Solr core. A `schema.xml` file is required with this data.

2.3. Incorporating an Algorithm into Seascape

Seascape can incorporate an algorithm written in any language with only a few steps. Seascape was developed with an emphasis on requiring very few changes to any existing code, as to not interfere with the developers research and development process. The majority of work required to incorporate an algorithm into Seascape involves preparing a small amount of additional support code. When incorporating a new algorithm into Seascape, the Seascape maintainer and algorithm developers must:

1. Prepare a release of the algorithm
2. Artifacts and Gitlab Structure
3. Creating a CI Schedule for your Algorithm
4. Adding Algorithm Tags

2.3.1. Prepare a Release of the Algorithm

The algorithm developer role prepares a release of the algorithm. Any algorithm being added to Seascape has the following requirements:

- Must be callable from a shell script.
- It must be able to accept the full path name to an image
- Some algorithms require external information or data for operation. An example of this might be a threshold value or a path to a file on disk containing weights for a trained machine learning style classifier. These data must be must be specifiable as command line arguments, in configuration files, or as environment variables. Any necessary files must be available within Seascape, either in Gitlab or Nexus, depending on the file size.
- The final output must be a set of JSON files following the Seascape schema detailed below

The algorithm may also utilize any additional parameters and outputs. The GitLab CICD pipelines which drive Seascape-VV communicate using the `IFILE` and `OFILE` environmental variables.

`IFILE` is populated with the path to a text file containing newline-separated paths to each image to be processed by the algorithm. For a baseline test, the list of images are described in the baseline test definitions JSON in Nexus. The test definitions file is discussed in Section 2.2.3. For a full evaluation, the test executor will select a portion of the entire dataset to run in each pipeline. The maximum number of data items in each pipeline is specified by the max_image CI variable in the `.gitlab-ci.yml`. Section 2.3.2 discusses the `runAlgorithm.sh` script.

`OFILE` is the path to a text file containing paths to each individual JSON result produced by the algorithm. The output paths are newline-separated, one path for each image provided in `IFILE`. Each JSON file will include a top level array of detection objects, each with the following keys:

```
featureID: <string>
    Note: Unique value across the all algorithm runs. This may be a UUID
    generated by the algorithm post processing code.

class: <string>
    Note: Class definition, can be as generic as ``target''

polygonPxl: <list of float pairs>
    Note: Polygon corners in (x,y), also known as (column, row),  pixel
    location. This should be a closed polygon (first/last point are
    the same)
```

An example of both the `OFILE` results text file and an individual image results JSON file can be found in Appendix G.

2.3.2. Artifacts and GitLab Structure

```
Seascape
└─Seascape-VV
  └─<Your Program>
    └─Algorithms
      └─<Your Algorithm Git Repository>
        ├─<algorithm specific code>
        ├─information.json
        └─runAlgorithm.sh
```

Figure 2-5. GitLab group structure of the Seascape algorithm-level. This folder structure provides the automated process framework for finding each artifact needed for algorithm benchmarking.

2.3.2.1. Create your algorithm git project

The Seascape maintainer is responsible for preparing the algorithm project folder in GitLab where the algorithm repository and other artifacts will be located.

1. In GitLab/Seascape/Sescape-VV/Your_Program/Algorithms, click New Project
    a) Project Name: <Your algorithm name>
    b) Project URL: <Your algorithm name>
    c) Click Create project

2.3.2.2. Required artifacts

Your algorithm git repository As described in Figure 2-5, each new algorithm repository to house algorithm specific code and artifacts.

information.json This JSON defines the name and version of the algorithm, and is used by the report generation tool when creating the report. The JSON file should have the following keys:

```
Algorithm: <string>
   Note: Name of your algorithm

Version: <string>
   Note: A moniker for the current version of the algorithm to use in the report
```

An example can be found in Appendix G.

runAlgorithm.sh This script is a wrapper for the algorithm. This artifact is created by the algorithm developer. This allows flexibility in the algorithm's language. The script is effectively the interface between the test harness and the algorithm itself. It calls the algorithm with the required inputs, specified by Seascape-VV, does any pre-processing on the data (for example file format conversion), provides any flags necessary to configure the algorithm, post-processes the algorithm output, if necessary (for example converting the algorithm output to JSON), and provides the resulting output to Seascape-VV. The script should do the following:

1. Activate any necessary environments (for example a pip or Conda environment)
2. Compile the source code if necessary to produce an executable file
3. Provide data to an algorithm via the `IFILE` environmental variable, which contains the image list as defined in TestDefinitions
4. Run the algorithm with the proper configuration on the image list, producing an appropriately formatted output
5. Export the path of each data results to the `OFILE` environmental variable environment variable
6. Deactivate environment, clean up, if necessary

An example script can be found in Appendix G. This function is performed by both the maintainer role, who is responsible for the creation of the algorithm GitLab structure, and by the algorithm developer role, who is responsible for providing the algorithm Git repository and required artifacts.

2.3.2.3. Optional artifacts

Any additional files which will, for example, to support environmental creation or allow for easier algorithm integration are welcome and encouraged. Any additional artifacts will be ignored by the Seascape system. Please note, if you include any additional artifacts which may utilize a GitLab runner, such as a GitLab CI/CD pipeline, please work with the Seascape maintainers to coordinate this.

2.3.3. *Creating a CI Schedule for your Algorithm*

In order to add an algorithm to the automated capabilities of GitLab CI CD, a new CI Schedule needs to be created in Seascape-Core for each new algorithm. This is a task to be performed by the maintainer role.

1. In GitLab/Seascape/Sescape-VV/Seascape-Core, click CI CD, click Schedules

    a) Click New Schedule

        i. Description: <Algorithm Name>

        ii. Interval Pattern: <Cron Formatted Time denoted how after to run>

        iii. Cron Timezone: UTC

        iv. Target Branch: master

        v. Variables:

        vi.

Variable	Value
algorithm_repo	<Your Project>/<Algorithms>/<Your Algorithm>
dataset_name	<the dataset name from Nexus/Data>
score_metric	<your algorithms score metric (iou,overlap,etc)>
target_list	<list of targets from dataset in Nexus/Data> (no https://)
max_images	<number of images per trigger per schedule run>
max_triggers	<number of triggers per schedule run>
max_pool_size	<??????>

2.3.4. *Adding Algorithm Tags*

Seascape-VV, specifically the test executor, relies on tagged commits to specify which algorithm version should be run through the Seascape-VV pipeline. The algorithm developer should tag a commit in order to communicate to the pipeline that this is the version to use during evaluation. Multiple tags can exist for separate commits. The following is how to tag an algorithm commit in GitLab:

1. In GitLab/Seascape, go to your algorithm page.

2. Select Repository > Tags > New Tag

    a) Tag name: <Your tag name>

    b) Create from: <specific commit hash> or master

Your tag must be created from the master branch in order to be run within the automated pipelines. Please note, test executor will only automatically evaluate algorithms with tags of suffix *_full* or *_baseline*.

REFERENCES

[1] Apache solr reference guide.

[2] DOCKER. Install Docker Engine on Ubuntu.

[3] GITLAB. GitLab CI/CD Pipeline Configuration Reference.

[4] GITLAB. GitLab Continuous Integration & Delivery.

[5] GITLAB. Install GitLab Runner manually on GNU/Linux.

[6] NVIDIA. Installation Guide.

APPENDIX A. Setting up Seascape From Scratch

In this section, we describe step-by-step how Seascape is set up from scratch. These tasks are performed by the Seascape maintainer. For most users of the Seascape system, there will already be an installed Seascape instance available for use, and these steps will not need to be followed.

> ⚠ All steps below involving Docker are optional. Not all networks allow the use of Docker. Please skip these steps as appropriate for your environment

Before these steps are begun, the following software programs must already be installed onto the system:

- Solr
- Banana
- Nexus
- GitLab

A.1. Setting up Seascape From Scratch in GitLab

This section houses the specific steps to set up a brand new Seascape instance. The ordering is important. Please follow the sections and steps in order. This is a task for the Seascape maintainer.

A.1.1. Create a user access token

This user token will be used in the repositories created below

1. In GitLab, click your profile picture (upper right hand corner) and click Settings
2. Click Access Tokens
    a) Name: seascape-vv
    b) Check "api"
    c) Check "read_user"
    d) Check "read_api"
    e) Check "read_repository"

f) Check "write_repository"

g) Check "read_registry"

h) Check "write_registry"

i) Click Create personal access token

j) A token string will be displayed at the top.

k) Record the value of the token for future use. This will be used for the access_token CI/CD variables later on

A.1.2. Create the Seascape group

The Seascape Group is the top-level gitlab object. It will house all other subgroups and repositories. Steps:

1. In GitLab

   a) Click Groups -> Explore Groups

   b) Click New Group

      i. Group Name: Seascape

      ii. Visibility Level: Private

   c) Click Create Group

   d) Click Settings (Gear Icon)

      i. Enter Description

      ii. Click Save changes

      iii. Expand Permission,LFS,2FA

         A. Uncheck "Allow users to request access"

         B. Check "Allow projects within this group to use Git LFS"

         C. Default Branch Protections: Fully Protected

         D. Allowed to create subgroups: Owners

         E. Click Save Changes

   e) Click Members

      i. add each member with their respective roles

A.1.3. Create the Seascape-VV subgroup

The Seascape-VV GitLab group is a subgroup of Seascape. It will house all other Seascape-Core and your projects algorithms needed for Seascape-VV. Steps:

1. In GitLab, browse to Seascape
    a) Click New Subgroup
    b) Enter:
        i. Group Name: Seascape-VV
        ii. Visibility Level: Private
    c) Click Create Group
    d) Add CI/CD Variables
        i. Click Settings->CI/CD
        ii. Expand Variables and Enter (uncheck Protected Variable for all):
        iii.

Variable	Value
access_token	<the user access token created above>
access_token_username	<the username associated with the access token>
assessor	correlation
dataset_repo	Data
docker_registry	???
gitlab_url	<your main gitlab url (no http://)>
nexus_url	<your main nexus url>
report_def_repo	ReportDefinitions
report_repo	Reports
score_metric	iou
test_def_repo	TestDefinitions
test_results_repo	Results
timeout_seconds	7200

A.1.4. Create a GitLab runner(s) for Seascape

A GitLab Runner is a Linux computer that will be set up and used to run the algorithms, calculate their performance, and create reports of results through the GitLab CI/CD mechanism. At least one is required, however, more is preferred.

This can be done on any machine of your choice. The specific instructions will be for Ubuntu 18.04

Steps:

1. Install GitLab Runner [5]

```
>> curl -LJO <https://Path/To/GitLabRunner.deb>
>> dpkg -i <GitLabRunner.deb>
```

2. Install Docker [2]

```
>> curl -fsSL https://get.docker.com -o get-docker.sh
>> sudo sh get-docker.sh
>> sudo usermod -aG Docker gitlab-runner
```

3. Start and Enable Docker

```
>> sudo systemctl start docker
>> sudo systemctl enable docker
```

4. Install Nvidia-Docker [6]

```
>> distribution=\$(. /etc/os-release;echo \$ID\$VERSION\_ID)
>> curl -s -L <https/To/nvidia-docker/gpgkey | sudo apt-key add - \
>> curl -s -L <https/To/nvidia-docker.list>
       | sudo tee </Path/To/nvidia-docker.list>
>> sudo apt update
>> sudo apt install -y nvidia-docker2
```

5. Install Ansible

```
>> sudo apt-get install Ansible
```

6. Install Vagrant

```
>> sudo apt-get install vagrant
```

7. Install VirtualBox

```
>> sudo apt-get install virtualbox
```

8. Install Git LFS

```
>> sudo apt-get install git-lfs
```

A.1.5. Register GitLab runner(s) to Seascape

This sections will register your newly created GitLab Runner(s) to Seascape.

> ⚠ Exercise forethought in choosing tag names for your runners. Avoid program- or hardware-specific names (unless actually necessary for a particular algorithm). Prefer name such as “has-nvidia”, “has-docker”, etc. Also avoid version tags, such as “has-python-3.9”. Generic names will make it easier to upgrade hardware or handle program renames.

Prerequisites:

1. Must have at least one gitlab-runner setup

Steps:

1. In GitLab, browse to the Seascape Group

   a) Click Settings (Gear Icon) -> CI/CD

   b) Expand Runners

      i. Under Setup a group runner manually

         A. Record the value of "Register the runner with this URL:"

         B. Record the value of "And this registration token:"

2. On the GitLab Runner host

   ```
   >> sudo gitlab-runner register
   ```

   a) Uses the following Settings

      i. Enter the URL from above

      ii. Enter the registration token value from above

      iii. Enter a description: <machine name>-shell

      iv. Enter tags for the runner: docker,nvidia-docker,shell-runner,seascape,seascape-shell,<machine name>-shell

      v. Enter an executor: shell

   ```
   >> sudo gitlab-runner register
   ```

   a) Uses the following Settings

      i. Enter the URL from above

      ii. Enter the registration token value from above

        iii. Enter a description: <machine name>-docker

        iv. Enter tags for the runner: docker-runner,seascape,seascape-docker,<machine name>-docker

        v. Enter an executor: docker

        vi. Enter default Docker image: alpine:latest

3. In GitLab, browse to the Seascape Group

    a) Click Settings->CI/CD

    b) Expand Runners

    c) For each runner

        i. Click edit (pencil icon)

            A. Check "Paused runners doesn't accept new jobs"

            B. (only shell runners) Check "Indicates whether this runner can pick jobs without tags"

            C. Click Save Changes

A.1.5.1. Setup the Seascape-Core repository

Seascape-Core houses the full capability of Seascape including the automation of testing.

Prerequisites:

1. Obtained seascape-core-master.tar.gz

Steps:

1. On Linux Box

```
>> tar -xvf seascape-core-master.tar.gz
>> cd seascape-core-master
>> git init
>> git add .
>> git commit -m "Initial"
>> git push --set-upstream
   <http/To/GitLab/Seascape/Seascape-VV>/Seascape-Core.git master
```

2. In Gitlab

    a) Browse to Seascape/Seascape-VV/Seascape-Core

    b) Create a Trigger

        i. Click Settings -> CI/CD

ii. Expand Pipeline triggers

A. Description: Performance

B. Click Add Trigger

C. Denote Token:

D. Denote Trigger URL (last line of curl command):

c) Add CI/CD Variables

i. Click Settings -> CI/CD

ii. Expand Variables and Enter (uncheck Protected Variable for all):

iii.

Variable	Value
algorithm_repo	ExampleAlgorithms/Huron
max_images	1
max_triggers	2
max_pool_size	10
performance_trigger_token	<token denoted above>
trigger_url	<trigger url denoted above>

A.1.5.2. Adding the Example Algorithm

1. In GitLab, browse to Seascape/Seascape-VV

a) Click New Subgroup

b) Enter:

i. Group Name: ExampleAlgorithms

ii. Visibility Level: Private

c) Click Create Group

2. Obtained Huron-master.tar.gz

3. On Linux Box

a) Copy over Huron-master.tar.gz

```
>> tar -xvf Huron-master.tar.gz
>> cd Huron-master
>> git init
>> git add .
>> git commit -m "Initial"
>> git push --set-upstream
     <http/To/GitLab/Seascape/Seascape-VV/ExampleAlgorithms>/Huron.git master
```

4. In Gitlab, browse to Seascape/Seascape-VV/ExampleAlgorithms/Huron

a) Click Repository->Tags

b) Click New tag

c) Enter:

   i. Tag Name: v1_baseline

d) Click Create Tag

A.1.5.3. Testing the Example Algorithm

1. In GitLab, browse to Seascape/Seascape-VV/Seascape-Core

   a) Click CI/CD -> Schedules

   b) Click New schedule

   c) Enter:

      i. Description: Huron

      ii. Uncheck Active

   d) Click Save Pipeline Schedule

   e) Click the Play button on the schedule you created

   f) Click CI/CD -> Pipelines

   g) Verify all Pipelines successfully complete

A.2. Putting Your Program in Seascape

Seascape can incorporate an arbitrary number of programs. A program is defined as an area that will hold multiple algorithms.

A.2.1. Create your program space in GitLab

1. In Seascape/Seascape-VV, click New subgroup

   a) Group Name: <Your Program Name>

   b) Group URL: <Your Program Name>

   c) Click Create Group

A.2.2. Add the Seascape harness to your program

1. In GitLab/Seascape/<Your Program Name>, click New subgroup
    - a) Group Name: Harness
    - b) Group URL: Harness
    - c) Click Create Group

1. In GitLab/Seascape/<Your Program Name>/Harness, click New project
    - a) Click Create blank project
    - b) Project Name: Performance
    - c) Project URL: Performance
    - d) Click Create Project
    - e) Click Settings->Repository
    - f) Expand Mirroring repositories
        - i. Git Repository URL: https://<your username>@<your gitlab url>/Seascape/Reference/Harness/Performance.git
        - ii. Mirror direction: Pull
        - iii. Authentication method: Password
        - iv. Password: <your gitlab password>
        - v. Check Overwrite diverged branches
        - vi. Click Mirror Repository
        - vii. Click the refresh button next to row created (circle arrows)
        - viii. Check default branch and protected branch settings, and set them to match the original repository, if necessary

2. Go back to GitLab/Seascape/<Your Program Name>/Harness
    - a) Click New project
    - b) Click Create blank project
        - i. Project Name: Reports
        - ii. Project URL: Reports
        - iii. Click Create Project
    - c) Click Settings->Repository
    - d) Expand Mirroring repositories

i. Git Repository URL: https://<your username>@<your gitlab url>/Seascape/Reference/Harness/Reports.git

ii. Mirror direction: Pull

iii. Authentication method: Password

iv. Password: <your gitlab password>

v. Check Overwrite diverged branches

vi. Click Mirror Repository

vii. Click the refresh button next to row created (circle arrows)

viii. Check default branch and protected branch settings, and set them to match the original repository, if necessary

e) Click CI/CD -> Schedules

f) Click New Schedules

i. Description: GenerateReports

ii. Interval Pattern: Custom 1 * * * *

iii. Cron Timezone: UTC

iv. Target Branch: master

v. Check Activate

vi. Click Save pipeline schedule

A.2.3. Add the Algorithms subgroup

1. In your program area, click New subgroup

a) Group Name: Algorithms

b) Group URL: Algorithms

c) Click Create Group

A.2.4. Add your algorithms

Please reference Section 2.3

APPENDIX B. Adding Algorithms via Docker

Seascape can also incorporate algorithms wrapped within a docker container. Algorithm Developers can use this capability to make their algorithms more portable, allowing them to ship the algorithm with pre-installed dependencies, instead of installing the dependencies at runtime. This appendix outlines the responsibilities of integrating a Dockerized algorithm into Seascape between the algorithm developer and Seascape maintainer. This section also assumes the Algorithm Developer interested in Dockerizing their application will already be familiar with the basics of docker.

B.1. Seascape Maintainer Responsibilities

B.1.1. Prerequisites

- Algorithm Repository has been created in Seascape as per Section 2.3.2
- Possesses the source code or Docker container for algorithm that meets algorithm developer requirements as per Appendix B.2.3
- Agreed upon Docker container tag provided by the Algorithm Developer
    - tag_name:_______________

B.1.2. Gather Information

1. Denote the Seascape Algorithm path
    a) Login into the Seascape and browse to your Algorithm Repo
    b) Look at the URL and Denote:
        i. algorithm_path: /seascape/seascape-vv/_______________
            - example: /seascape/seascape-vv/examplealgorithms/huron-docker
2. Get the gitlab URL:port
    a) In your Algorithm Repo
        i. Click Packages & Registries -> Container Registries
        ii. Click CLI Commands

iii. Look at the text of the login command, denote the gitlab_url_port as everything after "Docker login"

A. gitlab_url_port:______________

- example: cee-gitlab.sandia.gov:1234

B.1.3. Build and Push Docker Image into Seascape

This section outlines the best practice of this goal, acknowledging there are other ways to build and push. The Algorithm Developer may have provided you with a pre-built Docker container, or the algorithm source code and a Docker file. If pre-built, extract the image and skip to step 5.

1. git clone <Algorithms source repository>

2. cd into directory

3. Docker login <gitlab_url_port>

4. Docker build -t <gitlab_url_port>/<algorithm_path>:<tag_name>

   - example: Docker build -t cee-gitlab.sandia.gov:1234/examplealgorithms/huron-docker:v1.0.0

5. Docker push <gitlab_url_port>/<algorithm_path>:<tag_name>

   - example: Docker push cee-gitlab.sandia.gov:1234/examplealgorithms/huron-docker:v1.0.0

B.1.4. Verify Image in Seascape

This section outlines the best practice of this goal, acknowledging there are other ways to verify. After pushing your algorithm to gitlab, verify that it is there by:

1. Navigate to the algorithm repo

2. Click "Packages & Registries" -> "Container Registries"

3. Click on <algorithm_path>/ Root image

4. Verify your <tag_name> exists

B.1.5. Runners

Make sure that the machines designated as gitlab runners have Docker installed and accessible by the gitlab-runner user.

B.2. Algorithm Developer responsibilities

B.2.1. Gather Information

1. Work with the Seascape Maintainer to establish where the Docker image can be pulled from.
2. Obtain the appropriate login credentials to the container registry

B.2.2. Containerize the Algorithm

This can be done in many ways, however the container still needs to coordinate with runAlgorithm.sh to pass the appropriate values to the algorithm within the container. Tried and true methods of passing data to the algorithm include using Docker volumes as well as sending them over local host.

> ⚠ The one restriction to using Docker containers is to **not use bind mounts**. Bind mounts allow the Docker container to potentially leave behind file artifacts that the gitlab-runner user does not have permission to use. The runner will then fail all subsequent runs until the offending files are removed, likely with sudo/root access.

B.2.3. runAlgorithm.sh

The runAlgorithm.sh scripts needs to include the following steps, in addition to the requirements found in section 2.3.2.2:

1. Pull the Docker image from the algorithm container registry. Example:

   ```
           docker_tag=v1.0.0
           docker_image=cee-gitlab.sandia.gov:1234/seascape/seascape-
   vv/ examplealgorithms/huron-docker
           docker_image=$docker_image:$docker_tag
           echo "docker_registry: $docker_registry"
           Docker login -u ${access_token_username} -p ${access_token} $do
           Docker pull ${docker_image}
   ```

2. Run the Docker image
   - This can be done in many ways and it is up to the developer to decide how to ingest the contents of IFILE and output the results to OFILE
3. Save the OFILE results in $pwd/results.txt
4. Stop and remove the Docker container. Example:

```
        Docker rm \$(Docker stop \$Docker id) || true
```

5. Clean up
    - Deactivate and remove any conda environments
    - Clean up any other changes introduced by the Docker container

APPENDIX C. Seascape Core Development Process

The maintenance and expansion of the core Seascape capabilities should minimally impact the users of Seascape. The best method for the Seascape maintainers to accomplish this includes providing updates via the typical test, development, and production phases of software development. This will allow for adequate testing of the new additions which ensuring all users of Seascape are kept up to speed with any changes they must make.

In this section, we describe the Seascape Core development process. This process uses two Nexus Databases (Test and Prod) and contains three development phases: Development (Dev), Test, and Production (Prod).

C.0.1. Terminology

C.0.1.1. Nexus Databases

- Test
  - The database that the Seascape maintainer(s) will interact with during the Dev and Test phases
  - Dev branches and the Staging branch interact with this database
- Prod
  - The production database
  - The Master branch interacts with this database

C.0.1.2. Development Phases

- Dev
  - Uses the Nexus Test Database
  - Development branches, cloned from the Staging Branch, are utilized in this phase
  - The Seascape maintainer will determine the test(s) to prove their changes are correct
  - Nexus entries will be under <commit_hash>_<dev branch name> folders
- Test
  - Uses the Nexus Test Database

  - The Staging branch is utilized in this phases
  - Staging CI schedules are defined by the Seascape maintenance team in order to robustly test Seascape as a whole
  - Nexus entries will be under <commit hash>_staging folders
    * These folders are deleted periodically to ensure that the full processing of Seascape takes place on the Staging branch
- Prod
  - Uses the Nexus Prod Database
  - Nexus entries will be under <commit hash>_staging folders
    * Only changes that pass Test and are accepted by the Seascape maintenance team are incorporated into the Master Branch

C.0.2. Development Process

- Dev
  1. A GitLab Issue that documents the bug/enhancement/feature needs to exist. This can be create by a developer or the team.
  2. During a Seascape maintenance team meeting, the Seascape maintenance team will come to consensus about what issues need to be pursued. An issue will be assigned to a developer.
  3. The Seascape maintainer will then create:
     - A development branch cloned from the Staging branch
     - Dev CI Schedule(s) that points the development branch
  4. The Seascape maintainer will do all development on their development branch, while frequently pulling from the Staging branch continuously incorporate new changes from others.
  5. The Seascape maintainer will use their DEV CI Schedule(s) to test their changes.
  6. The Seascape maintainer will also create unit tests to cover their new/changed code.
  7. Once the changes have been completed, the Seascape maintainer will create a Merge Request into the Staging branch
  8. The Seascape maintenance team will review the Merge Request including the Issue and Source code. The Seascape maintenance team will either accept the merge or communicate that more work/changes need to be made
     - This includes unit test coverage and success of those tests

9. Once the Merge Request is accepted, the Seascape maintainer can move onto the Test phase.

- Test

  1. In the most common case, the Seascape maintainer will just need to wait for the all of the STAGING CI Schedules to complete successfully.

     - In some cases, the Seascape maintainer might need to create their own STAGING CI Schedule(s) to test their changes in the Test phase.

  2. If errors occur, based on the problem, the Seascape maintainer will either:

     - If the problem is large, remove the changes from the Staging branch, and go back to the Dev phases

     - If the problem is small, make the change to the Staging branch.

  3. Once all STAGING CI/CD Schedules complete successfully, the Seascape maintainer will schedule time with the Seascape maintenance team to show the results and discuss.

  4. If the Seascape maintenance team agrees that the change is appropriate and of quality, the Seascape maintenance team can decide to merge the changes into the Master branch.

- Prod

  1. a. This is the production version of Seascape. It will run the Master branch which is the most up-to-date, APPROVED code against the Nexus Prod Database.

APPENDIX D. FAQs

How do I pull data onto my local machine to train with? Use the function get_repo_contents found in the `nexusmanip.py` file within the Common configuration code repository, in Figure D-1.

```
Seascape
└─Common
   └─Ingest
      └─nexusmanip.py
```

Figure D-1. GitLab subgroup location of the `nexusmanip.py` function, which has functions to interact with Nexus.

How do I run the pipeline automatically? Assuming that there is a dataset and algorithm already in your program and Seascape has been configured by the Seascape maintainer to run your algorithm, there are two tasks to initiate an evaluation run.

The first task is to create a test definitions file and push it to Nexus. This is described in detail in Section 2.2.3. If there is already a test definitions file in Nexus to meet your needs, you can skip this step.

The next step is to tag an algorithm commit. This identifies an algorithm version for evaluated, as well as further articulates which dataset is the algorithm being evaluated against. Detailed directions can be found in Section 2.3.4.

Finally, as test executor runs on a schedule, please wait until test executor has executed the appropriate processes, evaluated the algorithm, and generated a report.

How do I run the pipeline by hand? Assuming that there is a dataset and algorithm already in your program, that Seascape has been configured by the Seascape maintainer to run your algorithm, and that there are test definitions and an algorithm commit tag in place, GitLab CI provides the ability to initiate a process by hand.

First, go to the GitLab process group you would like to run. For example, if you would like to re-generate a report by hand, go to the Reports repository, located in Figure **??**.

- Click CI/CD -> Schedules
- Click the play button

As a reminder, processes check for the existence of the artifact they create. Therefore, you need to remove that artifact from Nexus if you intend to recreate and replace it.

How do I access a report? The evaluation report is found in Nexus in the following location:

```
 Reports
└─Seascape
  └─<Your Program>
    └─Algorithms
      └─<Your Algorithm Name>
        └─<Your Algorithm Hash>
          └─<Your Test Definitions Name>
            └─report.html
```

Figure D-2. Location of the final report within the Nexus artifact structure.

How do I determine where to start with Seascape? The first step is to understand which role you have within Seascape. Roles are described in Section 1.3.1. Next, look in Section 2.1 to see which tasks are performed by each role. This section will point you to the correct tasks.

How do I know if my algorithm was incorporated into Seascape correctly? The easiest way to ensure an algorithm was correctly incorporated into Seascape is to look at the pipeline success indicator. If the pipelines all executed, next check if the correct artifacts were generated in Nexus, in reverse creation order. Ensure the final report was generated in Nexus. If the report is not there, verify that results were generated for each image within their appropriate location in Nexus.

APPENDIX E. Software Installation List

The following is a list of software packages and libraries necessary to stand up the Seascape system.

> ℹ The following assumptions and notes are made:
>
> 1. There is access to conventional software repositories such as PyPi or conda
> 2. These will be installed on a Linux-type operation system
> 3. Some programs inherently use libraries provided with the operation system. This document does not include those dependencies, for example, libraries installed with OS package managers such as yum, apt, etc.
> 4. There is a strong preference, but not a requirement, to have access to a RHEL satellite server

The following software programs are required to install Seascape:

- GitLab Version 12.4.2
- Solr 7.7.0
- jts-core 1.15.0
- Java 1.8.0_192
- Java Hotspot 64-Bit Server VM build 25.192-b2 mixed mode
- Banana 1.6.25
- Nexus 3.24.0-02
- Custom Seascape (Huron, optional) python and javascript files

In addition to the above mentioned packages, Python 3 must be installed on the system with access to the libraries listed in Table E-1 in order to run the custom Seascape (Huron, optional) software. Please note these package names and version are as they appear when utilizing the Anaconda Python package manager. There is no guarantee that the names or versions would appear identically as they would in another repository, such as Pip.

If installation of Huron, the test algorithm, is desired, the additional python packages are listed in Table E-2. Please note these package names and version are as they appear when utilizing the Anaconda Python package manager. There is no guarantee that the names or versions would appear identically as they would in another repository, such as Pip.

Table E-1. This table include all required python packages to install Seascape.

Name	Version	Name	Version
_libgcc_mutex	0.1	py	1.10.0
_openmp_mutex	4.5	pycparser	2.21
attrs	21.2.0	pyopenssl	21.0.0
beautifulsoup4	4.10.0	pyparsing	3.0.4
brotlipy	0.7.0	pysocks	1.7.1
bs4	0.0.1	pytest	6.2.5
ca-certificates	2021.10.26	pytest-cov	3.0.0
certifi	2021.10.8	python	3.9.7
cffi	1.15.0	python-dateutil	2.8.2
chardet	3.0.4	python-gitlab	2.4.0
coverage	5.5	python-slugify	5.0.2
cryptography	35.0.0	pytz	2021.3
cycler	0.11.0	readline	8.1
dill	0.3.4	requests	2.22.0
fonttools	4.28.3	rtree	0.9.7
idna	2.8	scikit-learn	1.0.1
iniconfig	1.1.1	scipy	1.7.3
jinja2	3.0.3	seaborn	0.11.2
joblib	1.1.0	seascape	0.1.0
kiwisolver	1.3.2	setuptools	58.0.4
ld_impl_linux-64	2.35.1	setuptools-scm	6.3.2
libedit	3.1.20210910	shapely	1.8.0
libffi	3.3	six	1.16.0
libgcc-ng	9.3.0	soupsieve	2.3.1
libgomp	9.3.0	sqlite	3.36.0
libstdcxx-ng	9.3.0	text-unidecode	1.3
markupsafe	2.0.1	threadpoolctl	3.0.0
matplotlib	3.5.0	tk	8.6.11
multidict	5.2.0	toml	0.10.2
multiprocessing-on-dill	3.5.0a4	tomli	1.2.2
ncurses	6.3	tqdm	4.62.3
numpy	1.21.4	tzdata	2021e
openssl	1.1.1l	urllib3	1.25.11
packaging	21.3	wheel	0.37.0
pandas	1.3.4	xz	5.2.5
pillow	8.4.0	yarl	1.7.2
pip	21.2.4	zlib	1.2.11
pluggy	1.0.0		

Table E-2. This table include all required python packages to install Huron.

Name	Version	Name	Version
blas	1	libtiff	4.0.10
bzip2	1.0.8	libuuid	1.0.3
ca-certificates	2019.10.16	libxcb	1.13
cairo	1.14.12	libxml2	2.9.9
certifi	2019.9.11	matplotlib	3.1.1
cloudpickle	1.2.2	mkl	2019.4
curl	7.55.1	mkl_fft	1.0.14
cycler	0.10.0	mkl_random	1.1.0
cytoolz	0.10.0	mkl-service	2.3.0
dask-core	2.6.0	ncurses	6.1
dbus	1.13.6	networkx	2.4
decorator	4.4.0	numpy	1.17.2
expat	2.2.6	numpy-base	1.17.2
fontconfig	2.13.0	olefile	0.46
freetype	2.9.1	openjpeg	2.3.0
freexl	1.0.5	openssl	1.0.2t
gdal	2.3.2	pcre	8.43
geos	3.6.2	pillow	6.2.0
giflib	5.1.4	pip	19.3.1
glib	2.56.2	pixman	0.38.0
gst-plugins-base	1.14.0	poppler	0.65.0
gstreamer	1.14.0	poppler-data	0.4.9
hdf4	4.2.13	proj4	5.0.1
hdf5	1.10.2	pyparsing	2.4.2
icu	58.2	pyqt	5.9.2
imageio	2.6.1	python	3.7.0
intel-openmp	2019.4	python-dateutil	2.8.0
jpeg	9b	pytz	2019.3
json-c	0.13.1	pywavelets	1.1.1
kealib	1.4.7	qt	5.9.6
kiwisolver	1.1.0	readline	7
krb5	1.16.1	scikit-image	0.15.0
libboost	1.67.0	scipy	1.3.1
libcurl	7.61.1	setuptools	41.4.0
libdap4	3.19.1	shapely	1.6.4
libedit	3.1.20181209	sip	4.19.8
libffi	3.2.1	six	1.12.0
libgcc-ng	9.1.0	sqlite	3.30.0
libgdal	2.3.2	tk	8.6.8
libgfortran-ng	7.3.0	toolz	0.10.0
libkml	1.3.0	tornado	6.0.3
libnetcdf	4.6.1	wheel	0.33.6
libpng	1.6.37	xerces-c	3.2.2
libpq	10.5	xz	5.2.4
libspatialite	4.3.0a	zlib	1.2.11
libssh2	1.8.0	zstd	1.3.7
libstdcxx-ng	9.1.0		

chapterTerminology

E.1. Git

Some basic Git terminology is useful in understanding and working with Git and GitLab (see section 1.3.2.1). These terms are used in the rest of the manual as defined here. For more information on Git, consult Stack Overflow or the Git web site at `https://git-scm.com/`.

Some key terms:

- Repository (also abbreviated as repo): a repository is a collection of files on a server
- Clone: cloning downloads the repository from the server
- Branch: a branch is a change history
- Merge: applying one branch's changes onto another branch
- Commit: staging changes to send to the main repository
- Push: sending changes to the main repository

Git has a few concepts that are important to remember:

- To add changes to tracking, use *git add <path>*. Multiple files can be added at once by specifying multiple paths or the path to a directory.
- Git doesn't automatically track all files in a directory; to add a file to change tracking, use *git add <path>*. Multiple files and directories can be added in the same way as done when adding changes from already-tracked files.
- To bundle changes and add them to the local change history, use *git commit*.
- To push committed changes to a repository, use *git push*.

E.2. GitLab

- Group: a group is a collection of groups or projects, analogous to a directory on a file system
- Subgroup: a subgroup is a group that is a child of another group
- Project: a project is a Git repository, along with things like pipelines, artifacts, etc.
- Artifacts: artifacts and outputs of pipelines, and can include (among other things) binaries, documentation, and test results

E.3. GitLab CI/CD

E.3.1. Runners

A *runner* is a machine running a worker process that accepts jobs from a GitLab server and executes the pipeline as defined in the configuration file (see section E.3.2).

Tags A tag is a descriptive string associated with a particular commit. Jobs can be assigned to particular machines with *tags* that have been associated to specific runners. This is generally done to ensure that necessary software or hardware, such as Docker or a particular type of graphics card, is present on the machine executing the pipelines.

Executor An *executor* is the method of execution on a runner machine. The most commonly used executor methods are:

- shell - Commands are run directly on the machine.
- Docker - A Docker container is started on the machine, and the pipeline is executed in the container.

E.3.2. Pipelines

A *pipeline* is an ordered collection of stages. A pipeline is defined in a YAML-formatted file commonly named *.gitlab-ci.yml*. This configuration file is discussed in more depth in Section E.3.2.

.gitlab-ci.yml The configuration of a CI pipeline is controlled by the configuration file.

Stages Stages are unordered collections of jobs. All jobs that are defined to run in a particular stage are run in parallel[3]. Stages are defined in the `.gitlab-ci.yml` file as a list:

```
stages:
  - build
  - test
```

Figure E-1. Stage Configuration in .gitlab-ci.yml

Stages can have any name, including spaces (although as a matter of good practice, you should avoid using spaces). Common names include:

- build
- test

- deploy
- clean
- publish

It is important to note that stages run *in the order they are defined*. In Figure E-1, the execution order would be *build*, *test*. Switching the order of the stages would produce wildly different (and profoundly more disappointing) results.

Jobs A job is the unit of work in GitLab CI. Each job is composed of several parts, with the most important covered here.

```yaml
cmake:
  stage: build
  before_script:
    - mkdir build && cd build
  script:
    - cmake ..
    - make
  artifacts:
    paths:
      - build
```

Figure E-2. Job Configuration in .gitlab-ci.yml

name Every top-level unknown key in an otherwise valid configuration file is parsed by GitLab CI as a *job*, with the key serving as the name of the job. In the example in Figure E-2, the name of this job would be *cmake*.

stage Each job must have a *stage* key. The value of the key is the stage to which the job is assigned.

before_script A job *may* have a *before_script* key. The value of the key is a list of shell commands to be executed before anything else takes place. If all jobs have the same commands in *before_script* sections, they can be abstracted into a top-level *before_script* section that applies to all jobs in the pipeline configuration.

script Each job must have a *script* key. The value of the key is a list of shell commands to execute. This is where you will place the command to build and test your algorithms.

artifacts Artifacts are outputs of jobs. These can include binaries, documentation, or other products of the shell commands in the *script* section. The *artifact* key can have several keys that control when artifacts expire (are no longer available to view or download), how they are stored, etc. For this manual, we will only cover the *paths* key. The *artifacts.paths* key has as its value a list of paths from the root of the repository to artifacts that are to be stored. Directories are valid artifacts, and will save the directory and all of its contents. In the example in Figure E-2, the *build* folder would be saved and downloadable.

E.4. Nexus

E.4.0.1. Repository

A *repository* is a structured collection of components (section E.4.2). Conceptually, it is organized like a filesystem on disk, with hierarchical folders. Each repository is associated with a *blob* (Section E.4.1).

E.4.1. Blob (Binary Large Object)

A *blob* or *blob store* is the raw storage format for Nexus. Typically, there is one blob per repository, in order compartmentalize changes to repositories or migrations.

E.4.2. Component

Components are the items requested by a build system, package manager, or other client to the Nexus instance. A typical example of this would be a PyPi package. Components can contain exactly one asset, as would be the case with an artifact like a PDF document. A PyPi component, on the other hand, contains a variety of assets, including the code, the setup file, examples of use, and many other files. A JAR component would contain at least a POM file and the compiled JAR files themselves. Components can also contain other components.

E.4.3. Asset

An *asset* is the fundamental item of storage. An archive file, PDF document, text file, or other type of file is an asset.

E.5. Solr

E.5.1. Core

A *core* is a single index and the related configuration and log files [1].

E.5.2. Index

A Solr *index* is a searchable data structure that returns documents matching queries[1]. A document is a collection of fields, which can be data or text references to where data can be accessed[1].

APPENDIX F. Miscellaneous Notes

If you are used to the version control system known as Subversion there are a few important differences to note when you are using Git:

- Multi-stage push: *svn commit* adds changes in tracked files to the change history and sends them to the main repository, whereas Git uses two commands to accomplish the same task:
  - *git commit* adds changes to a *local copy* of your repository
  - *git push* sends all the changes in your local repository to the server (or *origin*).
- When a Git repository is cloned, all branches are cloned as well, making your local copy a complete working copy of the repository
- *git clone* does equal *svn checkout*
- *git checkout* does *not* do the same thing as *svn checkout*
  - *git checkout* <*file*> updates a single file from the server repository
  - *git checkout -b* <*branch name*> creates a new branch
  - *git checkout* <*branch name*> switches to a branch

APPENDIX G. Example Code

Example code and files are available in:

```
Seascape
 └─Your Project
 └─Documentation
     └─Developers
         └─files
```

Figure G-1. GitLab subgroup location of example files and code.

APPENDIX H. Acronyms

This section lists all acronyms used in this document.

API	Application Programming Interface
ATR	automatic target recognition
CD	Continuous Development/Deployment
CI	Continuous Integration
CLI	command line interface
CPU	Central Processing Unit
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HDF5	Hierarchical Data Format 5
ICD	Interface Control Document
JSON	JavaScript Object Notation
ML	Machine Learning
NDA	non-disclosure agreement
OS	operating system
RD	research & development
REST	Restful State Transfer
SME	Subject Matter Expert
SSH	Secure Shell Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TIFF	Tag Image File Format
VLAN	Virtual Local Area Network
VM	Virtual Machine
VNC	Virtual Network Computer
VPC	Virtual Private Cloud
VPN	Virtual Private Network
V&V	validation and verification
HTTP	HyperText Transfer Protocol
XML	Extensible Markup Language
UUID	Universally Unique IDentifier

DISTRIBUTION

Hardcopy—Internal

Number of Copies	Name	Org.	Mailstop
1	John R. Dickinson	6773	0972
1	Jeffrey A. Mercier	6770	0980

Email—Internal [redacted]

Name	Org.	Sandia Email Address
Technical Library	1911	sanddocs@sandia.gov

Sandia National Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.