OGC Testbed-15

Scaling Units of Work (EOC, Scale, SEED)

Table of Contents

Publication Date: 2020-01-08

Approval Date: 2019-11-21

Submission Date: 2019-10-30

Reference number of this document: OGC 19-022r1

Reference URL for this document: http://www.opengis.net/doc/PER/t15-D021

Category: OGC Public Engineering Report

Editor: Alexander Lais

Title: OGC Testbed-15: Scaling Units of Work (EOC, Scale, SEED)

---

indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Chapter 1. Subject

This OGC Testbed-15 Engineering Report (ER) presents a thorough analysis of the work produced by the Earth Observation Clouds (EOC) threads in OGC Testbeds 13 and 14 in relation to the *Scale* environment. *Scale* provides management of automated processing on a cluster of machines and the *SEED* specification to aid in the discovery and consumption of a discrete unit of work contained within a Docker image. Scale and SEED were both developed for the National Geospatial Intelligence Agency (NGA) of the United States.

The ER attempts to explain how the OGC Testbed-13 [https://www.opengeospatial.org/projects/initiatives/testbed13] and OGC Testbed-14 [https://www.opengeospatial.org/projects/initiatives/testbed14] research results of "bringing applications/users to the data" relate to *Scale* and *SEED*.

Chiefly, while comparing the two approaches, the report identifies and describes:

- Opportunities for harmonization or standardization;
- Features which must remain separate and the rationale for this;
- The hard problems which will require additional work; and
- Opportunities which should be examined in future initiatives.

For developers, the ER constitutes a technical reference supporting the comparison of the two approaches, thereby enabling developers to make informed choices, understand trade-offs, identify relevant standards and clarify misunderstandings.

Chapter 2. Executive Summary

Multiple approaches for the distribution and deployment of workloads have been developed in recent years, and some of them have been the object of work performed in OGC Testbeds 13 and 14. This ER aims at comparing the work done in the EOC threads of those Testbeds, affecting relevant OGC standards, with the NGA Scale environment and its metadata specification "SEED".

Both approaches have in common that they encapsulate workloads in Docker containers, making them as independent as possible of the execution environment. More importantly, such Docker containers can be moved to nodes in the data center closest to the data or to other data centers altogether.

The added value of the assessed approaches is in the following areas:

- Discovery of available processes;
- Deployment of process instances where they are best suited, such as the node closest to the data or meeting other performance criteria best;
- Providing the necessary inputs to processes, possibly across nodes;
- Capturing process results as outputs, or as inputs for subsequent processes;
- Workflow orchestration, where multiple processes are chained together and process the respective output of their predecessor.

Achieving this functionality requires the awareness of available processes, data and resources. Discovery of processes requires a machine-readable manifest of the interfaces for a processor, including inputs in their supported formats, necessary processing resources to carry out their work and provided outputs in their supported formats. When this manifest is combined with Docker for encapsulating the program of the process, a reusable and transferable processing unit is created that allows a robust and scalable processing environment.

The OGC Web Processing Service (WPS) approach, discussed in the Earth Observation Cloud (EOC) threads of OGC Testbeds 13 and 14, does not directly prescribe how processes are to be executed, leaving the underlying technology flexible, while defining a clear and strict definition of the interface. Compatibility between different server and process implementations can be ensured by strictly verifying said interface. In terms of Application Programming Interface (API) design, WPS is aimed at interactive clients, based on direct selection of data and processes by the user, but also at scheduled bulk background processing.

On the other hand, Scale has exactly one deployment model aimed at a single data center with multiple nodes. Scale is not a standard or specification, but a specific implementation that defines its own particular interfaces. Being an implementation and runtime environment, Scale provides all the functionality needed to execute specific tasks and more complex workflows, including the scheduling of specific tasks on nodes and the provision of data. Scale is primarily intended as a data driven application that processes incoming data that is available in its workspaces. The starting point for processing is by default a new file in a folder, not a user request.

During OGC Testbed 15, Scale was evaluated in more detail, assessing similarities, benefits and drawbacks compared to the previously established OGC-based approach. The following main observations for the use of Scale in the context of processing are summarized below:

- The Scale model is designed for a very specific purpose with a narrow focus. With only one implementation to consider, fewer verification steps and less specification are needed for compatibility and interoperability. At the same time, Scale does not integrate with other systems directly. Using the API and wrappers, it can be integrated of course.
- On-demand processing, as needed for interactive clients, requires workarounds in Scale, as it is not the primary and intended mode of operation. Input and output data must be in Scale Workspaces, but there are no direct APIs provided for uploading or downloading such data. These can be added using other software, such as FTP or HTTP servers, or other means for file exchange.
- Experiments were carried out for wrapping calls to Scale jobs and workflows via OGC WPS. The inevitable similarity of needing inputs and outputs makes such a mapping straightforward when wrapping the simpler Scale APIs into OGC WPS calls.
- For declaring outputs, Scale prescribes specific files and locations to be created inside the Docker containers. The respective files are then picked up by the Scale runtime after a job has finished and before the Docker container is discarded.
- The SEED specification, intended as process descriptor for Scale jobs, encapsulates all necessary information about a particular process and bundles it directly with the Docker image's metadata. This approach allows detailed discovery and assessment of suitable Docker containers by retrieving only the image from the registry. There is no need for accompanying services or files containing metadata.

In summary, Scale is a very concrete runtime environment and implementation for distributed processing, while the OGC WPS standard is a specification allowing interoperable implementations from different vendors. The flexibility and detail of the interface specification of OGC WPS would allow issuing Scale requests as part of a WPS workflow, if desired.

Scale has a different overall approach to processing. This approach can be wrapped into OGC WPS-compliant interfaces. The reverse would be possible to some degree by integrating a WPS-compliant client in a Scale job, but would lack the stringent verification of inputs and outputs in subsequent steps.

Following the assessment and presented conclusions, the recommendations from this ER are as follows:

- **Consider adding metadata for jobs and workflows into Docker image metadata**
  The interface definition metadata embedded with the container, as done in the SEED specification, allows a wider and more automated discovery and reuse of process implementations. With the move to dockerized workloads, the process doesn't have to be hosted by a particular entity anymore but can instead be transferred to the environment closest to the data.
  Metadata that is embedded in the container helps with discovering, integrating and using such processes with the appropriate input validation.
- **Provide guidance on the mapping of SEED manifests to WPS process descriptions**
  The SEED specification does not depend on Scale and was designed with environments other than Scale in mind. Creating a best practice on how to use SEED-compliant containers in a WPS

environment, enabling their exploitation within OGC WPS workflows, would be an interesting possibility facilitating integration. SEED-compliant images could then be discovered and reused following the aforementioned semantics, but integrated into WPS-compliant workflows.

- **Consider using Scale as backend for EOC ADES instances**
  EOC ADES instances are responsible for deploying and managing processes in a particular environment. OGC WPS does not prescribe a particular implementation for the scheduling of work, actual deployment of processes or data transfer. Those tasks could be solved to a large extent by wrapping existing Scale functionality with an appropriate facade that implements the OGC WPS standards and mediates between WPS clients and the Scale environment.

OGC WPS-compliant implementations and Scale are not directly comparable in terms of scope, functionality or goals. This ER demonstrates that there are still synergies between the two approaches that should be explored, ultimately to the benefit of OGC WPS.

2.1. Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Contacts

Name	Organization	Role
Alexander Lais	Solenix	Editor
Paulo Sacramento	Solenix	Contributor
Juozas Gaigalas	George Mason University	Contributor
Ziheng Sun	George Mason University	Contributor

2.2. Foreword

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

Chapter 3. References

The following normative documents are referenced in this document.

- OGC: OGC 06-121r9, OGC® Web Services Common Standard (2010) [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2]
- OGC: OGC 14-065r2, OGC Web Processing Service 2.0.2 Interface Standard Corrigendum 2 (2018) [https://portal.opengeospatial.org/files/14-065r2]
- ISO: ISO/IEC 19510:2013, Information technology — Object Management Group Business Process Model and Notation (2013) [https://www.iso.org/standard/62652.html]

Chapter 4. Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard OGC 06-121r9 [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2] shall apply. In addition, the following terms and definitions apply.

- **Scale Job Type**

  A Scale `Job Type` is a template for a specific workload, defining the underlying Docker container, command and arguments to execute, expected inputs and outputs as well as resource requirements that are used for scheduling and allocation decisions.

- **Scale Job**

  A Scale `Job` is an invocation, i.e. an instance, of a specific `Job Type` on specific input data.

- **Scale Recipe**

  A Scale `Recipe` is a workflow that consists of various `Job Type` instances that consume the outputs of their predecessor as inputs.

- **Scale Recipe Execution**

  A Scale `Recipe Execution` is the invocation, i.e. an instance, of a specific Recipe with specific input data. Input data can be passed to one or more Jobs, which create outputs. Those outputs can then be passed to other jobs or considered as outputs of the overall Recipe when no further processing is desired.

- **Scale Result Manifest**

  A JSON document that describes the locations of outputs created by a specific Docker container, that underlies a `Job Type`, mapping named outputs to file paths within the Docker container. Scale can use the Result Manifest to identify and extract output files in order to provide them to subsequent `Job`s, or to store them in their destination Workspace.

- **Scale Workspace**

  A Scale Workspace is a file system location or AWS S3 Bucket, where `Job` inputs can be found or `Job` outputs can be placed upon successful execution. Scale does not provide an API for accessing the file data in a Workspace but can construct an externally accessible URL based on a configured base URL and the relative file path within the workspace. Such constructed URLs are also provided in the metadata of Jobs, allowing the retrieval of output data.

4.1. Abbreviated terms

- ADES Application Deployment and Execution Service
- AP Application Package
- API Application Programming Interface
- AWS Amazon Web Services
- BPMN Business Process Model and Notation
- CWL Common Workflow Language

- EMS Execution Management Service
- EOC Earth Observation Clouds
- ER Engineering Report
- FCA Federated Cloud Analytics
- ICT Information and Communication Technology
- JSON JavaScript Object Notation
- NGA National Geospatial-Intelligence Agency
- OGC Open Geospatial Consortium
- OWS OGC Web Services
- REST Representational State Transfer
- S3 Simple Storage Service
- WPS Web Processing Service
- XML eXtensible Markup Language

Chapter 5. Overview

This chapter provides an overview of the information laid out in this ER.

Chapters 1 and 2 contain the subject, executive summary and contributors of the ER.

Chapter 3 lists relevant references and standards.

Chapter 4 lists relevant terms and definitions.

Chapter 5, the present one, describes the structure of the ER.

Chapter 6 provides an overview of distributed processing approaches and the main challenges, briefly introducing the two approaches under examination in the ER and their key differences.

Chapter 7 contains an extended state of the art for the two approaches at hand, EOC and Scale/SEED, describing the work carried out in the EOC thread of previous Testbeds and describing Scale/SEED in some detail.

Chapter 8 assesses and summarizes the work carried out in Testbed-15.

Chapter 9 provides a conclusion and recommendations for future work.

Appendix A provides a revision history of this Engineering Report.

Appendix B contains the bibliography for the used reference documents.

Chapter 6. Overview of Distributed Processing Approaches

Datasets in the geospatial domain keep increasing in size and the algorithms applied for computation keep increasing in complexity. Clusters, Grid computing and other means of processing and distribution are increasingly being used for the processing of such large data sets.

In recent years, organizations started being able to rent processing power from cloud providers in a highly flexible fashion. This has enabled a wide array of tools and approaches, as well as a changed mindset, for the operation of large scale processing power when it is required. Tools, such as Cloud orchestrators, Docker, and Kubernetes, primarily add the ability to provision, prepare, control and contain runtime environments that are easy to re-create, are disposable and most importantly transferable to other execution environments (e.g. data centers).

The approaches discussed in this ER focus on bringing the algorithms and their execution closer to the data, avoiding the need to transfer very large datasets. This is achieved by packaging the programs running the algorithms in a standardized fashion, so they can be executed closer to the data, ideally in the same data center that stores the data in the first place.
Furthermore, chaining of different processes relying on multiple datasets and computations can be executed in different data centers and consolidated in yet another. Since the outputs of a processor usually reduce, refine and filter the amount of data when compared to the original inputs, only those new outputs then need to be transferred for consolidation.

The following sections briefly outline the challenges for improving the status quo in terms of transferability, compatibility and ease of use. This discussion is followed by a short summary of the approaches explored in previous OGC Testbeds.

6.1. Challenges for Improving Distributed Processing

Earth observation products and other geospatial data lend themselves well to parallel processing, where data is subdivided into pieces, such as tiles, that can be processed independently and merged together once the computation is complete. This is already possible with the aforementioned classical setup of a computational cluster.

Distributed processing nowadays also includes geographic distribution of data, e.g. agencies concerned with data for a particular region of a country, or the collaboration of various national agencies for processing at global scale. Even though the data may be in the same format and could be used by the same processor, transferring the data to a single location is costly and possibly inefficient in terms of required bandwidth, storage space and consequently the operator’s budget.

The primary challenges for improving on the status quo for distributed processing are in the following areas:

- **Bringing the algorithm to the data**
  EO Products are becoming increasingly larger, adding pressure on the required bandwidth and storage capacity in data centers. At the same time, products can often be stored or archived closer to the primary site of an interested party, such as an agency, scientific research center or

satellite operator. Processing the data at the location where it is stored reduces bandwidth and storage requirements.

- **Expressing and bundling a processor in a transferable way**
  Processing of geospatial data is done by computer programs, which may depend on a runtime environment dictated by the used programming language, additional libraries, calibration data and other auxiliary information. Deploying such programs and their required environment can become a complex multi-step process. Fortunately, the various tools provided by cloud computing infrastructures provide the fundamental means to create self-contained and compact bundles that encapsulate the program and necessary data in a transferable format.
- **Lifecycle control for execution, progress control, error handling and data I/O**
  Once the processing environment is transferred and in place, the ability to control, inspect and chain various processes together into the necessary workflow for a task is important. By defining a common interface for interacting with the application and its respective execution, setting up and controlling the workflow consisting of processors from different implementers in a single environment is possible.
  Processes usually work on specific types of data, focusing on a type of sensor, data representation, and often inherent properties of a specific platform. Accordingly, the suitability of a processor for a specific type of data can be determined based on the combined meta-information of input data and processor.

The most important aspect for the distribution and processing across agencies and data centers is the use of well-defined standards that specify data formats, supported processing capabilities, available data and available processing capacity.

Industry partners and agencies alike can provide the implementation of algorithms aimed at a specific result or field of research. Discovery features for processes allow interested users to find, assess, buy, retrieve and use such implementations in order to use them on their data or in their workflow. Solutions for discovery are a secondary but equally important aspect for the efficient use of distributed resources.

6.2. New Approaches to Distributed Processing

One of the primary challenges, as discussed above, is the encapsulation of an algorithm and the clear definition of inputs, outputs, error states and resource requirements. This definition and format must be created in a way that an execution orchestration system can invoke the algorithm by itself or in a workflow, ensuring compatibility and availability of input data as well as correct execution on the most suitable processing node.

Two approaches that address this challenge are discussed in this Engineering Report (ER). The two approaches share some of the underlying technologies but differ in their overall utilization and philosophy.

This ER considers the following approaches:

- The Transactional extension of WPS (WPS-T) [1] used in the EOC threads of Testbeds 13 and 14, featuring dynamic definition, upload and deployment of processors, together with a format for metadata describing the process and lifecycle.

- The *Scale* processing environment that aims at high-performance processing and dynamic workload distribution to various participating nodes. *Scale* [2] is supported by the *SEED* [3] metadata specification and was explored with a data center deployment as part of Testbed 15.

The approach taken for EOC (WPS and WPS-T) was assessed in OGC Testbeds 13 and 14, it is summarized in [4] for the process description and [5] for the execution orchestration environment. Consequently, this Engineering Report will summarize and highlight the aspects relevant for the comparison with the approach taken by *SEED* and *Scale* respectively.

The *Scale* Data Center (D145) deployed and provided as part of OGC Testbed 15 was used as a base to assess similarities or distinct qualities when compared to the EOC approach. Furthermore, this Engineering Report documents *Scale*'s overall setup and generic use with best practices, highlighting benefits, limitations or general points to consider when deploying into a *Scale* environment.

Both approaches are reviewed and compared in Chapter 7.

Chapter 7. EOC, SCALE and SEED

One of the goals of this Engineering Report is to document the exploratory work carried out in Testbed 15 and put it into relation with the results achieved in the Earth Observation Clouds (EOC) thread of Testbed 14. The goal is to identify commonalities, differences and synergies between the EOC approach and Scale/SEED, in light of the existing OGC standards for Processing.

In order to put the two approaches into context, the aforementioned commonalities in terms of technology and philosophy need to be discussed. The following section provides a high-level overview of both and closes with a mapping of concepts and the terms used in the respective approaches.

7.1. Earth Observation Clouds (EOC) Approach

The EOC thread of the OGC Testbed-14 project addressed among other topics the packaging, deployment and chaining of processes in a reusable manner.

The following elements in particular are relevant for this ER:

- OGC Testbed-14: Application Package Engineering Report [4]
- OGC Testbed-14: ADES & EMS Results and Best Practices Engineering Report [5].

The relevant concepts of that Testbed are the process orchestration (ADES & EMS) and the encapsulation of workloads into an Application Package that describes the inputs, outputs and other metadata for a particular process.

7.1.1. Application Package

The Application Package (WPS-T DeployProcess) defines a deployment descriptor for a WPS-compliant job into a WPS server.

This descriptor contains the following information about:

- The Process:
    - Inputs, including mime type or applicable schema;
    - Outputs, including mime type or applicable schema;
    - Context information, as needed;
    - Metadata, such as name, version, description, and so forth.
- The Execution Unit, which can be one or many of:
    - A specific Docker container to be executed. Based on the context information provided with the processor, the execution can be parameterized or tailored.
    - A specific workflow script that can be invoked on the processor directly.

During Testbed 14, the Common Workflow Language (CWL) [5] was used as the base implementation for creating workflows and to model the data flow through a processing pipeline.

Using the clear definition of inputs and outputs defined by each process, a workflow can be validated to ensure compatibility.

While CWL was used for prototype implementations [4], it is not mandated or prescribed. CWL is one of the feasible languages. Other experiments in Testbed 14 utilized the Business Process Model and Notation (BPMN) [6] [1], an XML based language, which allows modelling workflows and processing graphs. Various graphical editors for BPMN are also available, making the workflows easier to visualize, grasp and design. BPMN is published by the International Organization for Standardization as ISO/IEC 19510:2013.

A process in the "Application Package" is defined using the WPS-T descriptor, containing the metadata and referencing one or multiple Docker containers or workflow scripts, providing specific functionality and outputs for a well-defined set of input data and metadata types. An example Application Package manifest taken from Testbed 14 [4] is shown below.

Application Package (WPS-T DeployProcess) Example

```
{
    "processDescription": {
        "process": {
            "id": "GeomatysNDVIStacker",
            "title": "NDVIStacker",
            "owsContext": {
                "offering": {
                    "code": "http://www.opengis.net/eoc/applicationContext/cwl",
                    "content": {
                        "href":
"https://raw.githubusercontent.com/Geomatys/Testbed14/master/application-
packages/NDVIStacker/NDVIStacker.cwl"
                    }
                }
            },
            "abstract": "",
            "inputs": [
                {
                    "id": "files",
                    "title": "Input NDVI Image",
                    "formats": [
                        {
                            "mimeType": "image/tiff",
                            "default": true
                        }
                    ],
                    "minOccurs": "1",
                    "maxOccurs": "unbounded",
                    "additionalParameters": [
                        {
                            "role":
"http://www.opengis.net/eoc/applicationContext/inputMetadata",
                            "parameters": [
                                {
                                    "name": "EOImage",
                                    "values": [
                                        "true"
                                    ]
                                }
                            ]
                        }
                    ]
                }
            ],
            "outputs": [
                {
                    "id": "output",
                    "title": "NDVI Image",
                    "formats": [
                        {
                            "mimeType": "image/tiff",
                            "default": true
                        }
                    ]
                }
            ]
        },
        "processVersion": "1.0.0",
        "jobControlOptions": [
            "async-execute"
        ],
        "outputTransmission": [
            "reference"
        ]
    },
    "immediateDeployment": true,
    "executionUnit": [{
            "href": "images.geomatys.com/ndvis:latest"
        }],
    "deploymentProfileName":
"http://www.opengis.net/profiles/eoc/dockerizedApplication"
}
```

This Application Package example shows the deployment of a process encapsulated in a Docker container, referencing the profile `http://www.opengis.net/profiles/eoc/dockerizedApplication` and the Docker image name `images.geomatys.com/ndvis:latest` respectively.

An Application Package can also reference a workflow, as mentioned before. The snippet below demonstrates how a CWL workflow can be referenced.

Application Package fragment referencing a CWL Workflow

```
{
    [...]
    "executionUnit": [
        {
            "href": "https://raw.githubusercontent.com/spacebel/testbed14/master/cwl-
examples/multiSensorNDVIStacker_2collections/multiSensorNDVIStacker_2collections-
v4.cwl"
        }
    ],
    "deploymentProfileName": "http://www.opengis.net/profiles/eoc/workflow"
}
```

A CWL workflow will then contain various procedure steps to be executed. Those procedure steps can of course also be calls to other processes on the same or on a different WPS-compliant processing environment.

Compared to a SEED manifest, which is discussed in section 8.3, an Application Package provides more information and various modes of operation. A more detailed discussion of both formats is presented in section 7.4.

7.1.2. Specific Workloads (ADES & EMS)

Processes are usually integrated into workflows that consist of multiple steps. In the OGC environment, processes are traditionally exposed via WPS, which defines the interfaces for inputs and outputs and other metadata. Calling a process is then handled by another request that provides the required data or URLs to said data. WPS does not enforce any particular deployment or encapsulation of the processes.

In the EOC thread of OGC Testbeds 13 and 14 a series of tasks was executed to encapsulate processes into Docker containers. Docker containers are transferable to other environments and bring all of the necessary dependencies with them. Docker images, the base for Docker containers, can also be exchanged very easily via Docker registries that allow retrieval of Docker images to the right environment and processing node.

The role of the Application Deployment and Execution Service (ADES) [4] [7] in this scenario is to provide a runtime environment for dockerized processes, augmenting the underlying Docker engine. Most importantly, the ADES provides the WPS-compliant interface that allows deploying, controlling, monitoring and un-deploying processes according to the WPS requests and using the Docker engine. Additionally, the final provisioning of inputs and retrieval of outputs are in the domain of ADES.

The Execution Management Service (EMS) can be understood as a coordinator or orchestrator of a workflow that may be executed across different ADES instances. The prime example of such a workflow would be a deployment of ADES instances at different processing centers, which are made available for a unified workflow. Each process deployed on the respective ADES instance has the benefit of being close to the data hosted in that particular data center.

Complex and overarching workflows as described above will eventually require data to be transferred from one data center to another. The EMS is responsible for coordinating this data transfer and to continue the execution of a workflow once the input conditions are met, that is all of the necessary data for the next processing steps is accessible by the process.

7.2. Scale and SEED

Scale and SEED are both developed for the National Geospatial Intelligence Agency [8] of the United States.

Scale is an execution and task orchestration engine that utilizes Docker containers. The containers are provided with input data, execute their work and Scale extracts the resulting output data.

SEED is a specification that allows expressing a unit of work, its inputs, outputs and resource requirements in a generalized and reusable fashion. SEED was originally developed for use in Scale, but is agnostic to the execution environment.

Compared to the overall approach of a full WPS and WPS-T implementation, Scale and SEED are much more limited in scope and flexibility.

The following sections provide more details about Scale and SEED, demonstrating how they fit together and how they fit into the landscape of existing OGC standards.

7.2.1. Scale

Scale is essentially an execution engine that uses and orchestrates Docker containers to execute specific workloads. Scale provides a full Representational State Transfer (REST) API for controlling various processes, monitoring resource utilization, and managing workflows. By default, Scale executes configured workflows once data becomes available in a workspace.

The Docker containers represent the units of work, encapsulating all software that is necessary to execute a specific algorithm implementation. The use of Docker provides a simple and transferable means to package, annotate, distribute, reuse and retrieve specific algorithm implementations.

Scale is by default a data-driven platform. Scale monitors the files in its workspaces and allows the creation of triggers on the arrival of new files or metadata, which can then be used to trigger the execution of a series of algorithms on the data.

Algorithms are encapsulated as **Job Types**, which act as templates for jobs and define the required input data, possible auxiliary data and the expected outputs of the algorithm. Various jobs can be tied to a workflow, called a **Recipe** in Scale, which starts with input data and allows passing outputs of one job to the input of other jobs. Outputs can be defined as optional, and will be picked up by the system only when they were produced. Depending on the existence of such optional outputs, subsequent jobs are conditionally launched when their respective input data is available. This allows creating generally applicable recipes that execute common tasks on a variety of data and more specialized tasks depending on the effective data types of results.

Scale orchestrates multiple processing nodes (i.e. independent computers) and is responsible for providing the prerequisites to the Docker container execution. The data for specific workspaces is in a location accessible by all nodes, such as a shared network drive. This avoids moving the output data to specific nodes, as is done in other work orchestration engines such as Hadoop or Spark.

Resources of the respective nodes in terms of memory, CPU, and so forth are monitored and taken into consideration when scheduling tasks on a specific node.

Scale provides and supports two ways to define **Job Types**: via a simplified form that is specific to Scale; and via SEED, which is a specification attempt for generally reusable workloads. Both ways were explored during Testbed 15 and are described in this ER. The experience gathered is discussed in Chapter 8.

7.2.2. SEED

SEED is a specification of the **Job Type** description and metadata as utilized by Scale, but designed in a fashion that makes it useful and usable in other environments as well.

The main purpose of SEED is to enable discovery of Docker images that can be used in Scale and other systems. Discovery in this case includes a machine-readable specification of acceptable inputs and provided outputs, resource requirements and additional information that is useful for operators.

A SEED image has essentially the following characteristics:

- It is a Docker Image
- It follows a naming convention of `[name]-[version]-seed`, e.g. `orthorectification-1.2.4-seed`
- It contains a SEED manifest as Docker label (`com.ngageoint.seed.manifest`)

This information can then be used to query a Docker registry for images that end with `-seed` and contain the SEED manifest. Once that information is retrieved, the manifest can be further analyzed to determine whether that image contains a suitable task for the job at hand.

A support tool, the `seed-cli` is available for packaging an algorithm into an appropriate SEED-compatible Docker image.

7.2.3. Discovery of SEED Compliant Docker Images

The `seed-cli` tool mentioned in the previous section allows searching for and retrieving metadata about SEED compliant images from Docker registries.

In its current incarnation (`v1` and `v2` alike), the Docker Registry API does not provide means to search for the existence or content of labels on Docker images.

The SEED CLI works around this limitation by using the following approach [9]:

- Retrieve the names of all images in a particular repository, using the `_catalog` endpoint or search API on Docker Hub
- Filter all images that match the naming convention `*-seed`.
- For each of the retrieved images, query the tags and labels individually.

Once images are pulled into the local Docker engine, they can be queried and filtered for the existence of a label and its value, using the following command:

```
docker images -f label=com.ngageoint.seed.manifest
```

7.3. Commonality Analysis and Conclusions

Both discussed approaches have similar objectives and achieve their particular goals with components or elements for which a counterpart can be found in the respective other. However, a direct comparison is not entirely possible because of two main reasons.

1. From a strategic point of view, the EOC work aims at making use of OGC standards to define two main building blocks of an Exploitation Platform, the ADES and the EMS, as well as the main functionalities allocated to each of them. EOC also prescribes the OGC interface between the two, which is currently based on WPS(-T).
   Scale and SEED have no such aim and currently do not make use of OGC interface standards.
2. From a technical point of view, EOC has a strict requirement to be ICT and even platform-agnostic. All the addressed concerns are a level above the actual processing infrastructure. Due to their nature and target environment (a computer cluster), Scale and SEED instead explicitly address concerns at a lower level and necessarily closer to the ICT aspects.

Both approaches are to a large extent complementary, not competing or interchangeable. They can indeed work together and possible integrations are discussed in sections 8.5 and 9.3.

The remainder of this section provides a cursory glance and high-level comparison of the information handled by the different systems and the resulting possibilities for supported workflows.

The table below contains this mapping and provides an overview of the used terminology in both approaches.

Table 1. Conceptual Mapping between EOC and Scale/SEED

Purpose	EOC	Scale/SEED
Executable Unit Packaging	Application Package (AP)	SEED, Scale Job
Execution management	Application Deployment and Execution Service (ADES), Execution Management Service (EMS)	Scale
Process Chaining (Workflow)	Common Workflow Language (CWL)	Scale Recipe, Scale Strike

The EOC approach and its use of WPS and WPS-T for deployment of workloads is primarily an interface description and not an implementation. Consequently, implementations providing the interfaces should be interoperable. Scale is primarily a self-contained system that provides an end-to-end processing environment and takes control of all processing, providing all of the necessary functionality for it.

At first glance, the metadata processed by the EOC approach is more complex and extensive, compared to SEED and Scale. This complexity adds flexibility in the types of supported workloads and allows a very precise specification of supported inputs and expected outputs, beyond a simple claim to support a particular mime type. When processing EO data, the raw data products received from spacecraft usually differ significantly between missions and require very specialized processing. Being able to express the exact format that is supported is consequently crucial for this scenario.

Scale on the other hand provides a system that is easier to understand and requires less complexity for integrating new algorithms. A Docker image that encapsulates the process and produces the appropriate result metadata is already sufficient. Scale provides the input data and arguments for the configured application, and extracts results from the container autonomously. Error handling, status reporting and runtime management are fully covered by the Scale environment.

Chapter 8. Knowledge Gained in Testbed 15

SCALE was examined in detail in the OGC Testbed 15 Federated Cloud Analytics (FCA) thread. The thread tackled various topics, ranging from federation for processes and content exchange, to the exploration of new data center management software [10].

The primary aspect for documenting the knowledge gained in Testbed 15 in this ER was to capture emerging best practices, issues or constraints that have been observed while using the *Scale* environment in general or deploying processes or workflows specifically.

In order to support the Testbed 15 activities, the thread participants had to first become familiar with the overall concepts of *Scale* before starting to utilize the specific environment for exploratory work and interfacing with existing other components and OGC standards.

Based on the work carried out during Testbed 15, this section describes the observations, lessons learned, strengths and caveats of *Scale* as runtime environment and SEED as unit of work description and packaging standard.

8.1. The Scale Environment at George Mason University (GMU)

GMU deployed a *Scale* environment, spanning a total of 4 nodes, which could be used by Scale to execute workloads.

Overall, the standard environment for *Scale* deployments is a *Mesosphere DC/OS* Server, which provides access control, deployment orchestration for distributed applications, including those deployed in Docker. Mesosphere also provides convenient installers for various popular software packages that benefit from a distributed environment - among them also *Scale*.

At the time of writing, the latest available installer for Scale in the software repository for Mesosphere DC/OS was 5.4.0 and this version was used for the majority of the Testbed's duration. In collaboration with the Scale developers, GMU managed to install and upgrade Scale to version 5.9.7, which allowed testing of SEED-compliant images towards the end of the project.

Version 7.0.0 of Scale was nearing completion at the same time, but the Scale developers recommended to stay on the 5.x branch as it was the stable release.

8.1.1. Input Data and Workspaces

Scale jobs process data that is contained in **Workspaces**. These can be named folders, mounted network shares or AWS S3 buckets.

There is no API in Scale to upload or download files in a workspace. Instead it is expected that the mounted folders are accessible by other means. Fortunately, a base URL can be defined for each workspace, which is used by Scale to compute the absolute URL of a particular file in a workspace when it is exposed via HTTP. The full URL is then contained in processing results and can be used to directly retrieve result files.

For uploading files into the workspace as part of a processing workflow, typically the URL of a file or service, such as a WMS or WFS server, would be provided. Solenix developed a Scale job and corresponding Docker container that can download data via HTTP into a workspace and provide the downloaded files to other tasks. This allowed further exploring the capabilities and possibilities of Scale as runtime environment for distributed tasks.

8.2. Scheduling Work on Scale

Scale is primarily designed as a file-driven processing environment that executes its workloads on incoming files. Applicable jobs or recipes are scheduled based on the file's type, name or other criteria, which allows cascading work beyond jobs or recipes and to create more complex workflows purely by defining jobs for subsequent output types or file names. It is also possible to manually trigger workloads via the REST API. Both possibilities are discussed in the subsequent sections.

8.2.1. Automatic Workflow Execution with Ingest Jobs and Triggers

Scale recipes and jobs can be activated in two ways: automatic and manual. Recipes and jobs are activated automatically when Scale detects new input data for processing. Automatic processing requires configuring trigger conditions for data inputs. Jobs and recipes can also be triggered manually using the Scale RESTful HTTP API.

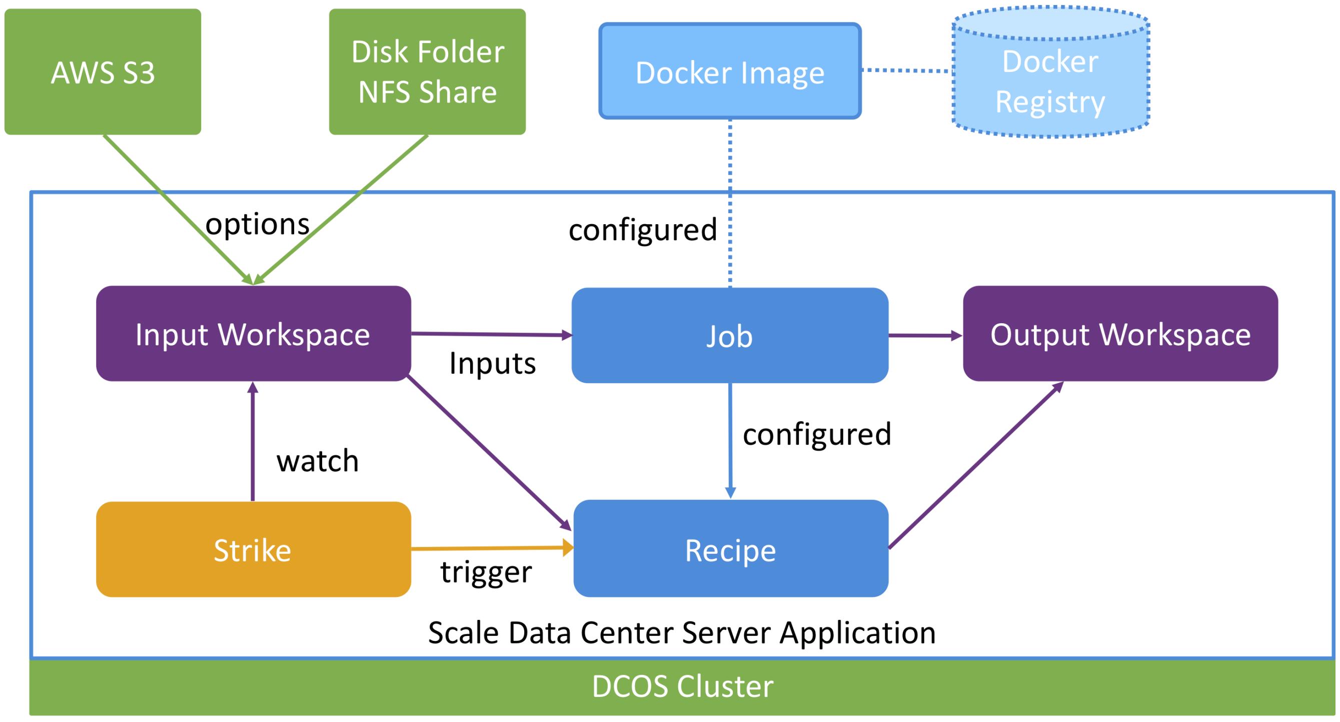

Figure 1: Scale Concept Overview

Automatic recipe and job execution has two steps. First, data files must be *ingested* into Scale, then recipes or jobs are *triggered* provided chosen conditions are met. Ingestion and triggers are configured using the Scale RESTful HTTP API. Ingestion is performed by special built-in Scale jobs called **Scan** and **Strike**. Scan is used to ingest pre-existing data files from a specified location. Strike is a continuously running job that watches specified locations for newly added files. Both Scan and Strike are configured to detect files that match specified name patterns. After they ingest a file, Scan or Strike will tag the new file with data type labels. Once a file is ingested and tagged, Scale is able to use triggers on that file.

There are 3 types of triggers: **input** (also called **ingest**), **parse** and **clock**. Input/ingest triggers are defined to match file path location and data type labels. When Scale detects that a new file was ingested it will find all triggers whose conditions match. Each trigger has a recipe type or a job type attached. Scale will launch a job or a recipe for each matching trigger for each newly ingested file. Parse triggers are configured to match results' manifest files that are created by jobs once they are done processing files. Using parse triggers, it is possible to chain jobs or recipes via output products of other jobs and recipes. Finally, clock triggers can be used to launch jobs or recipes at regularly scheduled intervals.

8.2.2. Manual Workflow Execution

Scale jobs and recipes can be launched manually using the RESTful HTTP API. The API user specifies job type id or recipe type id. They also provide input and output data parameters, such as input file name or output directory path.

In Scale version 5.9.7 the `POST /queue/new-job` and `POST /queue/new-recipe` API endpoints are used. In versions 6.0 and higher, the `/queue` API is deprecated - instead `POST /jobs` and `POST /recipes` are used.

Manually executed jobs or recipes respond with a descriptor, which also contains an ID. This ID can be used to query the status and to retrieve job results.

In order to demonstrate the different types of workflow outputs, in particular also simple outputs, such as single parameters instead of full files, another test job was created, which uses ImageMagick to provide statistical parameters as output. Scale jobs only support file-based results, while SEED-compliant jobs also allow JSON-based results, where individual keys of a JSON document can be mapped as simple results.

8.3. Anatomy of a Job

A Scale **job** is an abstract elementary unit of work - a single task to be performed on input data. A job produces some output data and a results manifest. Jobs are simple units of work designed to be chainable into workflows of work called recipes. The processing algorithm that a job performs is defined by a Docker image.

In Scale, jobs are defined by API objects called **job types**. A job type specifies the Docker image which can do the task and it defines input, output and configuration parameters that are supplied to the Docker container during job execution.

This definition object tells Scale which Docker image to use to execute the task, the resources required to run the container, the mounted volumes for the Docker container, optional environmental variables and other settings for the algorithm in the Docker image and expected error behavior. This definition also includes the **interface** specification.

The interface specification tells Scale that this job type takes specific inputs and produces specific outputs. The interface specification allows Scale to connect discrete jobs into workflows called recipes.

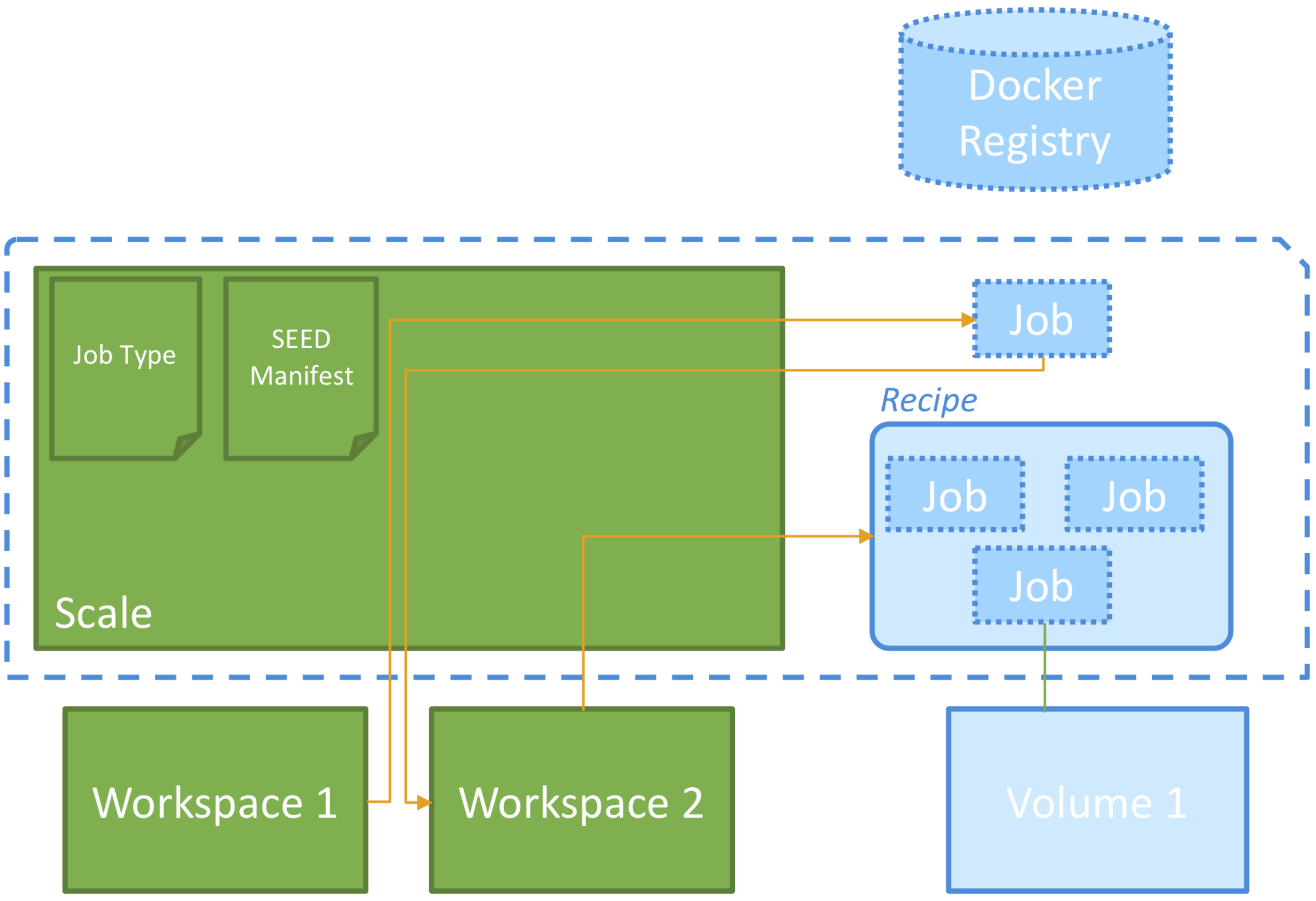

Figure 2: Scale Job Execution

Scale supports two formats for describing job interfaces. The original interface configuration and the new SEED-compatible interface specification. The SEED compatible format is only supported in versions 5.6 and higher. Some parts of the SEED specification, such as JSON based results, are only supported on Scale 6.x and higher.

Both the original and the SEED job interface specifications serve the same task, to define job inputs and outputs, but their syntax is slightly different. A SEED manifest can be packaged together with the container it is describing. This is discussed in section 8.7.

An example of a SEED-compatible job as result of executing a SEED-compliant job type is shown below:

seed.manifest.json

```
{
  "seedVersion": "1.0.0",
  "job": {
    "name": "curl",
    "jobVersion": "1.0.0",
    "packageVersion": "1.0.0",
    "title": "CURL Downloader (SEED)",
    "description": "A SEED compliant CURL downloader",
    "tags": ["curl"],
    "maintainer": {
      "name": "Alexander Lais",
      "organization": "Solenix Deutschland GmbH",
        "email": "alexander.lais@solenix.ch ",
        "url": "https://www.solenix.ch",
        "phone": ""
      },
      "timeout": 3600,
      "interface": {
        "command": "${URL} ${OUTPUT_DIR}",
        "inputs": {
          "files": [],
          "json": [
            {
              "name": "URL",
              "type": "string",
              "required": true
            }
          ]
        },
        "outputs": {
          "files": [
            {
              "name": "output_file_pngs",
              "mediaType": "image/png",
              "multiple": true,
              "pattern": "*.png"
            },
            {
              "name": "output_file_csv",
              "mediaType": "text/csv",
              "pattern": "*.csv",
              "required": false
            }
          ],
          "json": [
            {
              "name": "num_files",
              "key": "numFiles",
              "type": "integer"
            }
          ]
        },
        "mounts": [],
        "settings": []
      },
      "resources": {
        "scalar": [
          {
            "name": "cpus",
            "value": 1
          },
          {
            "name": "mem",
            "value": 64
          },
          {
            "name": "sharedMem",
            "value": 64
          },
          {
            "name": "disk",
            "value": 512,
            "inputMultiplier": 4
          }
        ]
      },
      "errors": [
        {
          "code": 1,
          "name": "error-name-one",
          "title": "Error Name",
          "description": "Error Description",
          "category": "data"
        }
      ]
    }
  }
```

The **manifest** follows the SEED specification [https://ngageoint.github.io/seed/seed.html].

The manifest contains the following information:

- **seedVersion**: Declares the applicable version of the SEED specification. At the moment there is only version 1.0.0.
- **job**: Contains job metadata, similar to a Scale job
- **errors**: Defines the meaning for Docker process exit codes during failure
- **resources**: Cpu, memory and disk resources required for this job
- **interface**: Specifies how this job will consume and produce data
- **interface.command**: A string passed to a Docker container at runtime via the Docker `CMD` interface
- **interface.inputs.files**: A list of input files
- **interface.inputs.json**: A list of keys passed in a JSON object for the job
- **interface.outputs.files**: A list of output files
- **interface.outputs.json**: A list of keys returned by the job in a JSON object
- **interface.mounts**: Optional volumes to be mounted into the Docker container
- **interface.settings**: Environmental variables passed to the Docker container

When the job runs it must produce output files that are specified in the interface, some of which may be declared as optional. Two additional files must also be written to the root of the output directory contained in the Docker container:

- `seed.outputs.json`: Containing JSON output keys and values from SEED **interface.outputs.json**
- `results_manifest.json`: Listing output files produced and their paths. This file is only required when defining a Scale job, as the SEED manifest contains all expected output file name patterns.

8.4. Definition of Workflows via Scale Recipes

Jobs in Scale can be combined into a larger processing workflow, called a Recipe. The Recipe is essentially a wrapper for a group of Jobs that pass their outputs to subsequent jobs. When a job creates multiple outputs, one or more subsequent jobs can consume outputs in parallel.

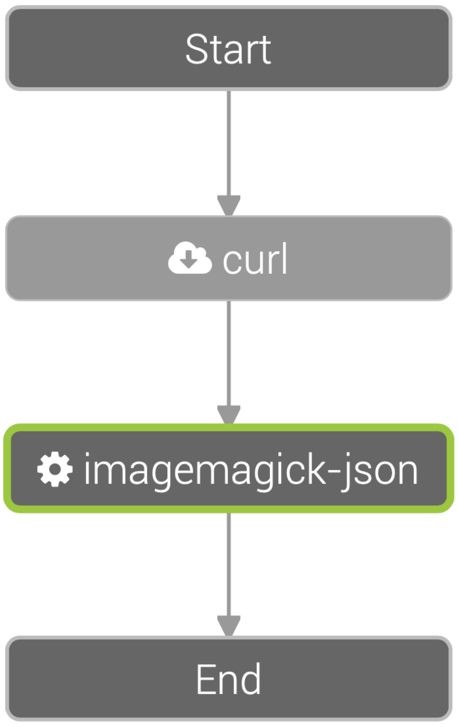

Figure 3: Scale Recipe Graph

In addition to Jobs, a Recipe defines inputs and outputs itself, which can be passed on or taken from the containing jobs. Recipe-level inputs are defined on the **Start** node and allow distribution of inputs to jobs, to define static parameter values passed to a job and aggregated outputs provided by different jobs. Recipe outputs can be bound via the **End** node respectively.

8.5. Integration into WPS Processes and Workflows

With the demonstrated ability to trigger individual jobs and recipes, Scale's capabilities can be wrapped to be utilized in a WPS server as well.

The following steps for integration were demonstrated during Testbed 15:

- Execute individual jobs, with status monitoring (started, done)
- Execute recipes and capture their output.

For the execution via WPS it was not relevant whether the job was defined as a Scale job or via a SEED manifest.

The integration of *Scale* workloads into WPS is discussed in detail in the Federated Cloud Analytics ER [10] of OGC Testbed 15.

8.6. Identified Limitations and Challenges

Testbed 15 presented the first opportunity for European players to explore the Scale environment in the context of an OGC Testbed. This posed some challenges particularly during setup, which are briefly summarized below:

- Input data must be in workspaces, which are managed by the Scale environment.
  - Adding files to the workspace requires configuration outside of Scale or a job that places data into the workspace.
  - For result retrieval, the workspace can be exposed via HTTP. A Base URL can be provided as configuration to each workspace, which allows Scale to construct appropriate URLs, which are placed into the job result information.
- The default installation of Scale in Mesosphere DC/OS does not support persistent databases. When the container hosting Scale is recreated, the data is also removed. The recommendation is of course to set up persistence appropriately.
- Installation instructions were outdated and have been extended and updated in response to requests and questions raised in this activity. The developer team of Scale was responsive and helpful via their Gitter channel.
- Only file-based results can be provided in Scale jobs and SEED jobs with Scale versions older than 6.x. Creating an additional file as job output that contains arbitrary metadata in JSON format proved to be a suitable workaround.
- Different versions of the Scale API place various of the utilized tasks into different endpoints. Scale 5.9.x supports version `v5` and `v6` of the API, allowing a smooth transition to the new `v6` API, while reducing the use of the now deprecated `v5` API that is planned to be removed in Scale 6.x and 7.x. In general, the deprecation of API versions seems rapid. On one hand this makes the software simpler, more robust and easier to maintain, while on the other hand backwards compatibility cannot be guaranteed for longer time spans.

8.7. Use of SEED in Testbed 15

During the majority of the Testbed 15 activity, Scale was available in version 5.4.0, which as of yet does not support SEED. Scale was later updated to 5.9.7, which does support SEED. SEED is not critical for the understanding of how Scale works, as it is another means to define the metadata for job types and to discover appropriate Docker containers containing the job’s logic.

In order to assess the potential and utility of SEED-compliant workload packaging, the aforementioned containers (for cURL and ImageMagick) were augmented with SEED manifests and SEED-compliant output metadata artifacts, which were created in addition to the standard `results_manifest.json` required for regular Scale Jobs.

SEED has been tested initially with the `seed-cli`, which allows running SEED-compliant tasks locally. The `seed-cli` tool allows building, inspecting and running SEED compliant images, searching for images in Docker registries and various other functions surrounding SEED images. Following those initial tests, the SEED compliant images were also deployed on the Scale instance provided by GMU.

The behavior of jobs was in line with the expectations discussed in section 8.3.

Chapter 9. Evaluation, Conclusion and Recommendations for Future Work

The main goal of this Engineering Report was to discuss, compare and assess the OGC WPS-based approach adopted in previous Testbeds in their EOC threads with the capabilities provided by the Scale environment and the SEED packaging standard. Both approaches are similar in various points but differ in others. The following section provides an evaluation, summarizing the results for the different criteria and highlighting particular strengths or weaknesses of the respective approach.

Following the evaluation, subsequent sections provide an overall conclusion and recommendations for future work.

9.1. Evaluation Criteria and Results

This section provides an overview of similarities and differences as well as strengths and weaknesses for those two approaches. In order to provide a systematic and fair assessment, a set of evaluation criteria was selected, summarizing points of similarities and differences between both approaches. Each criterion is presented with a short description of its purpose and impact, followed by the applicable comments for EOC vs. Scale/SEED.

For the purpose of this section, the OGC WPS standard as used in the frame of the Testbed 13 and 14 Earth Observation Clouds (EOC) threads' work is referred to as "EOC".

- **Algorithm Encapsulation and Packaging**
  Defines how easy it is to create or adapt a program to run in the desired environment.
  - EOC allows a variety of implementations to be wrapped into an Application Package descriptor. Docker-based images are supported and can be used in combination with a workflow description language. It is up to the user and execution environment to define whether a new process will spawn a new container or use a provided service that happens to be deployed as container.
  - Scale/SEED are strictly focused on disposable Docker containers, which provide the implementation and are instantiated on demand for each job execution. The container for a job is created, mounting a volume for the input data and the appropriate entry point and communicates the availability and location of results via a file that is extracted from the container by Scale.
- **Workflow Definition**
  Describes how a workflow - that is the chaining or branching of multiple processors -, can be specified and how outputs of the predecessor can be used as inputs for the next processor.
  - EOC allows the declaration of workflow scripts. The exact supported language or format depend on the execution environment, but different ones are possible. It is expected that future versions of the standards will define a preferred, or even mandatory, workflow language that must be used.
  - Scale defines workflows as "Recipes", which act as a "compound job" that also provides inputs and outputs. Recipes then contain chained jobs that either take a predecessor’s output as input, or an input that has been provided to the recipe. Accordingly, job outputs

can be declared as recipe outputs. A recipe can be used in place of a job to achieve more complex workflows, but following similar semantics. Recipes are declared as JSON objects, binding specific jobs and inputs as well as their outputs to recipe inputs and outputs, without additional logic.

- **Flexibility of Packaged Algorithms**
  Defines which types of programs can be encapsulated, and whether limitations apply.
  - EOC uses dockerized processes with defined interface for inputs, which contain all of the necessary data or locations where data can be retrieved. Additional logic can be provided with the workflow language that ties different processes into a larger workflow.
  - Scale and SEED allow packaging arbitrary algorithms in Docker containers. Auxiliary data can also be provided via volume mounts to containers directly and does not need to be provided as input.
- **Error Reporting**
  Considers the handling of errors, error reporting and behavior when errors occur
  - EOC uses the standard error reporting mechanisms of WPS by providing an exception report. Capturing and creating this report is the responsibility of the WPS runtime environment.
  - Scale and SEED use the return code of a process to determine possible error conditions. The return code is mapped to an error description that can then be shown to the user and is propagated with a job’s status.
- **Process Discoverability**
  Determines how processes can be discovered, assessed, evaluated and utilized.
  - EOC requires processes to be registered with the WPS server. Processes can then be discovered via WPS. Dynamically deployed workloads are registered on demand with an appropriate manifest, which is to be provided in addition to the Docker image name.
  - Scale jobs are defined with similar constraints as EOC jobs, i.e. the manifest is provided separately from the Docker image. SEED-compliant containers can carry their job description in the SEED manifest that is attached to the Docker image’s metadata.
- **Platform Independence**
  Defines whether the execution environment requires a specific platform or runtime environment to function, and how well it can adapt to or utilize other environments.
  - EOC primarily defines a series of interfaces, calling conventions and metadata formats, which are then used by the implementers of a compliant system to the standards. EOC allows any type of service and container to be used. There is a specific profile for dockerized workloads, but it is not mandatory. Execution targets may include workflow scripts to control the execution engine, which again may involve dockerized workloads but is not limited to them.
  - Scale/SEED require the use of Docker and heavily rely on Mesosphere DC/OS for the container orchestration, i.e. the scheduling, distribution, creation, monitoring and teardown of Docker containers used for execution.
- **Main mode of operation**
  Considers the designed mode of operation, even when other modes can be achieved as well.

  ◦ EOC is primarily request-driven. A client that wants data processed will usually issue the processing request, which initiates the provisioning of data and process containers, execution of the algorithms, collection of output data and provision of said output data to the client, e.g. via download link.
  ◦ Scale/SEED are primarily data-driven systems more akin to a batch processing system. The primary mode of operation is the monitoring of a workspace for the availability of files or metadata matching a specific type. When data is available, registered jobs or recipes are then launched automatically, usually placing the outputs into a workspace again. Cascading processing can thus be achieved based on the availability of data alone, without the use of explicitly scripted workflows.

9.2. Conclusion

The EOC approach and the Scale/SEED approach have been developed for a similar set of tasks: the efficient processing of large amounts of data with user provided algorithms. Both approaches have embraced the use of dockerized workloads as means to define reproducible, disposable and easily transferable execution environments for such algorithms.

EOC

- EOC is in an experimental stage with different approaches that have not been established as best practices for implementations.
- EOC is more flexible and allows different vendors to contribute elements to the overall systems. APIs and interfaces are more versatile, allowing a wider spectrum of workloads.
- EOC supports more flexible and more complex workflows due to the use of a workflow language.
- EOC is designed for interactive and non-interactive processing alike.

Scale

- Scale is an operational system with a more narrow focus aimed at mostly unsupervised and automated processing
- Workflows can be achieved via recipes and by cascading the production of files that trigger subsequent jobs
- Interactive processing can be achieved with workarounds but is not the primary focus of Scale.
- SEED and Scale jobs provide a more simplistic way to define inputs and outputs and don’t provide means for more complex input data validation.
- Scale is tied to Mesosphere DC/OS and Docker as execution environment, while EOC does not strictly prescribe an execution environment.

9.3. Recommendations for Future Work

Following the assessment, evaluation and experiments with the Scale software and its strengths, the points below were identified as potential inspiration for the next generation of OGC WPS APIs and for the possible integration of OGC WPS-based processing with Scale jobs or use of the Scale execution environment:

- **Metadata for jobs and workflows in Docker image metadata**
  The SEED specification allows embedding the interface definition in the Docker image’s metadata. Embedding the interface definition allows a wider and more automated discovery and reuse of process implementations. This is particularly important to avoid the mismatch when metadata and Docker image are maintained separately. With the move to dockerized workloads, the process doesn’t have to be hosted by a particular entity anymore but can instead be transferred to the environment closest to the data.
  Metadata that is embedded in the container only requires a Docker registry and helps with discovering, integrating and using such processes with the appropriate input validation.
- **Provide guidance on the mapping of SEED manifests to WPS process descriptions**
  The SEED specification is designed to be agnostic to the execution environment and does not depend on Scale. Processes for WPS and for Scale are similar in scope, and integrating support for SEED-compliant processes could be interesting for WPS-compliant servers. By creating a best practice on how to wrap and use SEED-compliant containers in a WPS environment, SEED-compliant images could be discovered and reused following the aforementioned semantics, but integrated into WPS-compliant workflows.
- **Consider using Scale as backend for ADES instances**
  ADES instances are responsible for deploying and managing processes in a particular environment. OGC WPS does not prescribe a particular implementation for the scheduling of work, actual deployment of processes or data transfer. Those tasks could be solved to a large part by wrapping existing Scale functionality with an appropriate facade that implements the OGC WPS standards and mediates between WPS clients and Scale environment. Other alternatives instead of Scale would be to use Mesosphere DC/OS directly, or utilise the more complex and more flexible Kubernetes.

Appendix A: Revision History

Table 2. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
October 30, 2019	A. Lais	1.0	all	issue for submission as pending to OGC TC Meeting
May 29, 2019	A. Lais	.1	all	Initial Engineering Report

Appendix B: Bibliography

1. Pross, B., Cauchy, A.: OGC Testbed-14: WPS-T Engineering Report. Open Geospatial Consortium, http://docs.opengeospatial.org/per/18-036r1.html (2019).
2. National Geospatial Intelligence Agency: Scale Web Site, http://ngageoint.github.io/scale/, (2019).
3. National Geospatial Intelligence Agency: SEED Specification, https://ngageoint.github.io/seed/seed.html, (2019).
4. Sacramento, P.: OGC Testbed-14: Application Package Engineering Report. Open Geospatial Consortium, http://docs.opengeospatial.org/per/18-049r1.html (2018).
5. Sacramento, P.: OGC Testbed-14: ADES & EMS Results and Best Practices Engineering Report. Open Geospatial Consortium, http://docs.opengeospatial.org/per/18-050r1.html (2018).
6. Pross, B., Stasch, C.: OGC Testbed-13: Workflows Engineering Report. Open Geospatial Consortium, http://docs.opengeospatial.org/per/17-029r1.html (2018).
7. Gonçalves, P.: OGC Testbed-13: Application Deployment and Execution Service ER. Open Geospatial Consortium, http://docs.opengeospatial.org/per/17-024.html (2017).
8. National Geospatial Intelligence Agency: National Geospatial Intelligence Agency Home Page, https://www.nga.mil/, (2019).
9. National Geospatial Intelligence Agency: Discovering SEED Images in a Docker Registry, https://ngageoint.github.io/seed/seed.html, (2019).
10. Gonçalves, P.: OGC Testbed-15: Federated Cloud Analytics Engineering Report. Open Geospatial Consortium, http://docs.opengeospatial.org/per/19-026.html (2019).