University of Victoria

Simplified Execution Engine

SEE, It works.

Requirements Document

Version 0.9

Document Produced for

SENG 371 Software Evolution. Spring 2019

Table of Contents

1 Introduction

1.1 Purpose

The goal of this document is to convey to the reader the design of the Simplified Execution Engine (SEE) to make available a multi-cloud, scalable, and user friendly service for performing research and analyzing of large datasets. SEE mainly focuses on earth observation data, but also encompases a generic base that would enable adaptation for different types of data.

1.2 Project Scope

The project scope encompases scientists, researchers, and anyone with interest in analyzing large datasets. This project offers a solution to complex configurations of computation resources, accessing and indexing datasets, and configuring source code for execution on a computation platform.

SEE bids farewell to the dark ages of downloading and unzipping datasets. Instead of bringing the large amounts of data to the execution environment, SEE brings your code into the cloud where the data already exists. This will grant faster data access, scalability with computation on the data, and providing features most wouldn't dare to implement to run their code.

1.3 Glossary of Terms

Term	**Definition**
AWS	Amazon Web Services. A cloud platform offered by Amazon.
BYOA	Bring your own algorithm.
CLI	Command line interface.
EoD	Execution on data.
FR	Functional requirement
Git	This is a source control management tool.
SEE	SEE being described by this document, Simplified Execution Engine.
SVN	Subversion. This is a source control management tool.
Users	Actors who will use the platform.

	Researchers, scientists, and people with a knack for computation on large datasets.
Workspace	A space in which a users code is contained. A user can have many workspaces, each with their own configuration and codebase.
VCS	Version Control System, such as Git.

1.4 Overview

This document contains all the information necessary to build SEE being described here. Listing all functional and nonfunctional requirements.

2 Overall Description

2.1 Product Perspective

As cloud services become more pervasive in the world of computation, a mechanism needs to be described to allow researchers, and scientists to fully utilize these cloud services. Our product positions itself to fill that exact void; SEE bridges the complexities of managing deployment and data access, and wraps it in a friendly user interface. This allows for users to perform data processing jobs without needing to know how to work the CLI.

2.2 Product Features

SEE allows its users to upload their code to the platform. Once the user's code has reached the platform it is ready for execution on all datasets available to that user.

2.3 User Classes and Characteristics

This section outlines some use cases that SEE must be designed to handle.

- User manually uploads their code for a quick execution.
- User pulling their code from a VCS platform and configures their execution.
- Organization publishes new data to a platform supported by our system.

2.4 Operating Environment

SEE is expected to be deployed on a cloud system such as AWS, as these platforms offer a resizable cluster deployment. This will fit the platforms need for availability, as the cloud provider will enable SEE to quickly scale up to meet researcher demands.

3 Functional Requirements

3.1 Overview

This section covers the different features that SEE will offer to the user.

3.2 Bring your own Algorithm

The user must be able to upload their own source code to SEE. Uploading code to SEE is the first step to getting starting, therefore it is essential to not restrict how it gets there. And the usage of VCS allows for quick access to existing codebases, and all the perks that VCS bring for software development which will enable power users.

FR 1 SEE must provide the user with a mechanism to upload files.

FR 1.1 SEE must provide the user with a mechanism to organize their files in a directory hierarchy.

FR 1.2 SEE must allow the user to drag and drop files into the window.

FR 2 SEE must allow the user to pull source code from Git, SVN, and Hg source code management hosts.

3.3 On-Platform BYOA Code Editing

The user must be able to edit their code that exists on SEE through a code editor within SEE. This avoids the annoyance of editing code on local machines and refetching the code to verify and validate changes.

FR 3 SEE must provide an interface for the user to edit their code.

FR 3.1 SEE should include a spelling check in the code interface.

FR 3.2 SEE should include syntax highlighting for files relating to the kind of code being edited.

FR 3.3 SEE should allow for the same editor to open and display images.

FR 4 SEE must provide a mechanism to preserve the changes made to their code on SEE.

FR 4.1 SEE must allow changes made to code to be integrated into an existing VCS.

FR 4.2 SEE must allow the user to download their code after being edited on SEE.

3.4 Triggered Execution

Triggers would allow for the execution of code on newly added data. The execution of code would work much like database triggers work; except instead of a tuple update in a database the STAC catalog would have a pending update. With that pending update code could be run before or after the update. For example,the code that is run before could be some logging function, or to make a measurement to see how the new data affects the current datasets. The code that is run after the update would also be highly useful as the code is queued for execution as soon as the data is made available, and there is no need to do some form of polling with scheduled execution.

FR 5 SEE must provide a configuration for execution of code when certain events happen.

FR 5.1 SEE must provide the option to execute code before an event occurs.

FR 5.2 SEE must provide the option to execute code in parallel with the event that triggered it.

FR 5.3 SEE must provide the option to execute code after an event occurs.

3.5 Scheduled Execution

Scheduled Execution would allow for SEE to enable researchers to automatically run jobs. Scheduled Execution could be done to run a job overnight, or to run jobs periodically on datasets that are frequently updated. This solves the problem that arises with triggers and datasets that update frequently.

FR 6 SEE must provide a configuration for execution of code when certain time conditions are met.

FR 6.1 SEE must allow the user to choose specific dates.

FR 6.1.1 SEE must offer weekly based recurrence.

FR 6.1.2 SEE must offer monthly recurrence

FR 6.1.3 SEE must allow recurrence based on a number of days that have elapsed.

FR 6.2 SEE must allow the user to choose specific times.

FR 6.2.1 SEE must offer times by default in 15 minute windows, but provide the option to select any minute specific time.

4 External Interface Requirements

4.1 Overview

This section covers interfaces that SEE will need to interface with. This includes support for various cloud providers, data storage providers, and source control management providers.

4.2 Cloud Providers

The end product should be fit to be interoperable between any mainstream cloud providers available. However, given the differences between how these providers operate, support will start with AWS and will roll out with more alternatives. SEE should attempt to include some form of customization for cloud providers that are not mainstream. This would make SEE more maintainable in the event of an API change, and would also allow for easy addition for support to more cloud providers. Coincidentally this also helps to avoid, if not prevent vender lock-in.

4.3 Data Storage

The data storage as it currently stands, is tied quite heavily into what cloud provider is being used. Data storage is tied heavily to the cloud platform, so the same requirements in the previous section apply here.

4.4 Source Control

Due to the nature of SEE as a BYOA styled systems, the users must have source control integration. SEE would need to support both Git, SVN, and eventually Mercurial in order offer the service to the most users. However, if a user used another provider they would still be able to upload their codebase manually.

The integration of source control would provide SEE with ways to quickly push, manage, and test changes. This would also allow for code that was changed through SEE to be committed back and saved as part of their version control history.

5 Non-Functional Requirements

5.1 Overview

This section outlines the requirements of the project that are not features. This covers the way the user experiences SEE. Some of these can only be measured qualitatively but some can be measured both qualitatively and quantitatively. Measuring methods will be defined in the next revision of this document.

5.2 Usability

Usability is key to being able to offer the platform to the largest number of potential users. As researchers and data scientists shouldn't be expected to know how to configure deployments and access to datasets. Part of usability will be designing useful signifiers so that the perceived affordances are aligned with the actual affordances.

5.3 Security

SEE will be a potential target for malicious code, which is why it is essential that each users code is run in isolation. The code must not modify datasets or infrastructure, and the execution environment must also be cleaned up after execution.

5.4 Power Usability

With all systems you will have highly technically inclined users, and SEE should allow them to use the same system to do more. Thus, SEE should offer advanced configuration for the users to tweak workspace settings.

5.5 Performance

SEE should offer high performance computing through the workspaces.

5.6 Responsiveness

The user interface should be smooth and responsive. To reduce any confusion that could arise from delayed feedback.