D9.8 Specification Report

Grant Agreement nr	780470
Project acronym	SAUCE
Project start date (duration)	January 1st 2018 (36 months)
Document due:	December 31st 2020
Actual delivery date	December th 2020
Leader	DNEG
Reply to	William Greenly - wmg@dneg.com
Document status	Submission Version

Project funded by H2020 from the European Commission

Project ref. no.	780470
Project acronym	SAUCE
Project full title	**S**mart **A**sset re-**U**se in **C**reative **E**nvironments
Document name	D9.8 - Specification Report
Security (distribution level)	Public
Contractual date of delivery	December 31th 2020
Actual date of delivery	December 31th 2020
Deliverable name	Specification Report
Type	Report
Status & version	Version for Peer Review
Number of pages	24
WP / Task responsible	DNEG
Other contributors	-
Author(s)	William Greenly - DNEG
EC Project Officer	Ms Adelina Cornelia Dinu - adelina-cornelia.dinu@ec.europa.eu
Abstract	This document provides a comprehensive report of the some of the key work packages and deliverables covered by the SAUCE project
Keywords	Search, Framework, Smart, Reusable, Asset, Classification, Transformation, Contribution, Descriptor, Vocabulary
Sent to peer reviewer	Yes
Peer review completed	Yes
Circulated to partners	No
Read by partners	No
Mgt. Board approval	No

Document History

Version and date	Reason for Change
1.0 22-11-20	Document created by William Greenly
1.1 21-12-20	Version for peer review
1.2	Final version for submission to EC

Table of Contents

1 EXECUTIVE SUMMARY

This document provides a complete specification report for a number of aspects of the SAUCE project in particular the Smart Search Framework. It should be considered as an asymmetric companion to D4.4, the Smart Search Prototype, serving to provide the non-functional characteristics and qualities of the Smart Search Framework.

We begin by reaffirming the objectives, providing background and defining the relationship with other work packages. Following on from this we provide a high level overview of the Smart Search Framework and explain how it is split into a number of related, but independent constituent parts.

We proceed to provide a detailed explanation of each of these constituent parts, the architecture supporting them along with reference implementations developed and delivered as part of the project. Whilst each of these parts can be deployed and adopted in isolation, we also allude to how they can work together, complimenting one another.

Finally, we provide information and insight into how many of the deliverables and outputs of the framework provide opportunities and enable other research and development, along with additional ways they can be integrated with industrial partners, consortia and institutions.

2 BACKGROUND

This deliverable is the specification report for work packages 2, 4, 5 and parts of 7, covering the search and transformation frameworks and aspects of storage. It builds upon a number of other deliverables, detailed in section 3.4, and represents the complete and comprehensive technical specification for the smart search framework and transformation framework.

This specification report is based upon actual technical deliverables which are referenced and cited throughout the document and can be made available for inspection or use at any point. There is nothing hypothetical or untested that is included in this specification.

3 INTRODUCTION

This chapter provides an overview of the specification report, re-establishes the goals and objectives of the specification and describes the relationship between the specification and various other deliverables.

3.1 Main objectives and goals

The main objectives of this deliverable are to:

- To provide an architectural overview of the search and transformation framework and its constituent parts
- To provide a detailed description of each of the building blocks along with different deployment options
- To provide guidelines for interoperability, collaboration and extension of framework components
- To provide suggestions for further development and continued research and development

3.2 Methodology

This document provides complete, comprehensive and detailed specifications for every component of the search and transformation framework. This is consistent with readmes and specifications contained within individual component repositories.

This document provides executable examples and references to source code repositories with all the outputs of the project. It should be noted that the GitHub repositories listed in this document are private and require access and will return 404's if accessed by an unauthorised user.

3.3 Relationship to Self Assessment

This deliverable has no direct relationship to the self assessment, the culmination of which is provided in D8.5, but does relate to the dissemination plan, and is integral for the dissemination and proliferation of the project and its associated deliverables.

3.4 Relationship to Other Work Packages

This deliverable provides the final specification report for the following deliverables :

- D2.3 Report on modules, transformations and APIs for framework
- D4.1 Smart Search Framework POC
- D4.2 Tools to validate and upgrade assets
- D4.3 Semantic Labelling Toolbox
- D4.4 Smart Search Prototype
- D5.2 Initial demo of tools to transform asset representation
- D5.3 Basic framework to enable asset transform
- Work Package 7, Asset Storage

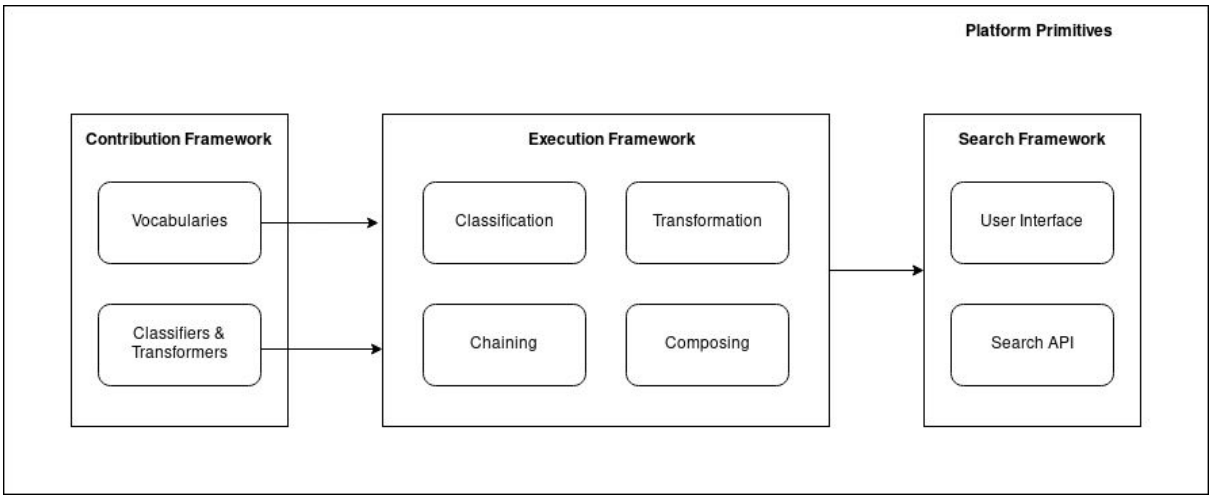
4 Technical Overview

The Search and Transformation Framework is split into a number of constituent parts, all of which are separate and independently deployable. These are listed below:

- Platform Primitives
- Contribution Framework
- Execution Framework
- Search Framework

Each part can be developed, deployed and operated independently and in isolation. They each provide value individually and collectively, so a user can pick and choose which parts to adopt in an isolated and iterative way, facilitating pragmatic business and commercial integration.

Furthermore, as described in D4.1, Smart Search Framework POC, for each framework, a modular development pattern has been followed (libraries, distributions, solutions) ensuring that even subsystems and subcomponents can be distributed and reused in different ways.



The diagram above displays the relationship between the frameworks and constituent parts and can be summarised as follows. The Contribution Frameworks provides a means for individual contributors to develop and share vocabularies, classifiers and transformers in a consistent and interoperable way. Vocabularies, classifiers and transformers are deployed into the execution framework, where their capabilities can be advertised for classification, transformation and implicit and explicit chaining. The outputs of the execution framework, along with available transformation options, are ingested into the search framework and made available to the end user via a search user interface, demonstrated in D4.4 Smart Search Prototype. All the aforementioned frameworks depend on a variety of subsystems for network, security, storage and compute, which are provisioned as platform primitives.

5 Platform Primitives

All the frameworks and components require a common and abstract set of components and subsystems to provide the following core capabilities:

- Networking, both ingress and egress
- Security
- Compute
- Storage

Throughout the course of this project, these capabilities have been generalised and defined as the platform primitives, and can be provisioned through an abstract and extensible framework.

This framework makes extensible use of a gitops pattern of provisioning and deployment and leverages the following dependencies across the gitops lifecycle:

- Terraform: for provisioning and infrastructure as code
- Git: for source control and target state

The general lifecycle for provisioning and deployment can be summarised as follows:

1. The latest master branch is checked out from the git repository
2. Provisioning and configuration can be tested locally before commits using the following commands. This runs exactly the same process as runs on continuous integration

```
circleci local execute --job plan \
-e GOOGLE_CREDENTIALS="$(cat $SERVICE_TOKEN_PATH)" \
```

3. Terraform validates the execution plan locally
4. Commit and push changes to the git repository master
5. The continuous integration server runs terraform to compare the execution plan against the current target state in the git repository
6. The continuous integration server runs Terraform to apply the differential to the target
7. The new target state is committed as the current target state in the git repository

This doesn't deviate from a standard Gitops workflow. All the projects listed in the subsequent sections follow this approach in order to plan, apply and track platform primitives infrastructure and configuration.

5.1 Networking

The following projects are responsible for networking

https://github.com/sauce-consortia/terraform-vpc
https://github.com/sauce-consortia/terraform-dns

These projects both contain Terraform manifests, that when applied to infrastructure, will provision and configure a number of network primitives including a virtual private network and DNS. It should be noted that DNS is also provisioned as part of compute, detailed in section 5.3, but that this is default for an interservice communication in Kubernetes. Any additional public or private DNS should be provisioned in the terraform project detailed above.

For the purpose of the demonstrator, public DNS was provided for the user interface, the search API and a number of classifiers and transformers in the execution framework

5.2 Security

Security is managed in the same project as the networking projects

https://github.com/sauce-consortia/terraform-vpc

This configures the various IAM policies for different resources across the platform, namely access control to various resources. In the instance there are broadly two types of access control:

1. Individual access control to resources
2. Application access control to resources

Both access control categories leverage the same OAuth Security model, implementing different flows.

Accessing the user interface

Individuals that want to access the user interface must have valid DNEG GSuite accounts. Upon accessing a protected resource i.e the user interface, they will be prompted to authenticate with their GSuite credentials. A authorisation code is then issued, and this is used to authorise the user and provide the user with an access token and refresh token. This follows the standard OAuth flow.

Accessing the API's

Access to API's, namely the search API and classification and transformation framework, are secured with client credentials. Client credentials are issued to third parties and these are used to authenticate and authorize 3rd party applications. In this instance, this is an essential part of the Flix integration.

5.3 Compute

Compute is provided via Kubernetes, a powerful and mature Linux container management system. This is setup, provisioned and configured by the following project:

https://github.com/sauce-consortia/terraform-k8s

This provides resources and compute required to run the execution and search framework. This project also bootstraps the kubernetes cluster with internal networking, service discovery and a bastion for deployments

5.4 Storage

The following project provides storage and persistence spanning a number of cases:

https://github.com/sauce-consortia/terraform-gcr

This configures and enables a number of different storage components, namely cloud storage and network file systems for a number of applications and systems listed below:

- Persistent disk for the Graphstore
- Persistent storage for containers, pods and services
- Asset storage

Integrating and Using Flix as Asset Storage

The aforementioned project provides storage and persistence for both applications and containers running in the compute platform (Kubernetes) along with general purpose storage external from a specific application context. One of the objectives of the project was to demonstrate the integration between the Flix asset storage and the Search and Transformation Framework and a number of integration points were demonstrated. One of these was the ability to use Flix as persistent storage for classification, transformation and search. The project below demonstrates how flix is integrated as a platform primitive into the SAUCE framework enabling access across a number of contexts:

https://github.com/sauce-consortia/flix-fuse-server

This project achieves a number of things, namely:

1. Integrates the Flix file system with the persistent storage, enabling both physical and application level sharing of data across systems
2. Installs a Flix application server and user interface into the compute cluster allow a Flix client to run and access storage and provide a fully fledged asset management system in the SAUCE Framework

There are recommendations later in this document for ways this can be further developed and extended.

6 The Contribution Framework

In D4.1 we described and demonstrated a number of core concepts of the Smart Search Framework, with particular focus on how contributors can share vocabularies for describing visual effects domains and subdomains, and how domain experts can share classifiers and transformers for use within the SAUCE framework. This has been further extended and sophisticated over the course of the project to facilitate publication and deployment of the aforementioned components.

6.1 Vocabularies

The contribution framework builds upon this and provides a means for contributors to not only write and develop vocabularies, but also to deploy and publish them into the framework so that they can be used by classifiers and transformers and extended and leveraged by other contributors.

D2.3 prescribed a new asset model for search and reuse and this was formalised in D4.1 and extended in D4.2, D5.2 and D5.3. This core vocabulary has been published and can be found below:

https://vocabularies.sauce-project.tech/core/core.ttl

This vocabulary provides core bindings for a number of other components and is published and deployed from the project below:

https://github.com/sauce-consortia/extensions-core

This provides an archetype for all other vocabulary contributions and a boilerplate has been provided, along with documentation, for contributors to follow in order to develop, deploy and publish further vocabularies. This can be found below:

https://github.com/sauce-consortia/extensions-boilerplate

Additionally a reference implementation of this archetype is also available and this demonstrates the ability to publish and deploy extension vocabularies.

https://github.com/sauce-consortia/extensions-images

This project proves that we are able to use the supplied boilerplate to publish and deploy a vocabulary representing concepts and terms from OpenImageIO.

6.2 Classifiers and Transformers

In D4.1 and D5.3 we demonstrated how partners could contribute classifiers and transformers into the SAUCE framework. This was exemplified by a number of reference implementations. Over the course of the project, as a result of a number of contributions from partners and developers, it was necessary to develop, improve, enhance and extend these concepts and provide a robust means to develop, publish and deploy classifiers and transformers.

The initial proof of concept leveraged similar patterns to those for publishing vocabularies, namely providing archetypes and boilerplates that could be used to develop interoperable and consistent classifiers and transformers. This included a number of common units of currency such as a base image and uniform interface and set of methods for executing a classification or transformation. Subsequently, informed by requirements, use cases and demand from contributors, this has been extended to incorporate the following additional features:

- Image/container initialisation hooks
- A reusable cache
- Polyglot runtimes
- Teardown and postprocessing hooks
- Asynchronous and synchronous behaviours
- Webhooks, callbacks and publish URL's

The project below provides a boilerplate and archetype for creating, publishing and deploying classifiers and transformers:

https://github.com/sauce-consortia/classifiers-base

A number of reference implementations are also available that leverage and utilise this archetype. One of these is a general purpose classifier that can be used as a common classifier for multiple asset types:

https://github.com/sauce-consortia/classifiers-generalpurpose

Over the course of the project a number of partners contributed classifiers and transformers for different types of asset, leveraging the boilerplate and base images. These are listed below:

A Wordnet Enricher *(DNEG)*
https://github.com/sauce-consortia/sauce-action-wordnetenricher

An ImageIO Enricher *(DNEG)*
https://github.com/sauce-consortia/sauce-action-imagelabelextraction

A Google Vision Enricher *(DNEG)*
https://github.com/sauce-consortia/sauce-action-sampleenricher

An Image Type Classifier *(DRZ)*
https://github.com/sauce-consortia/sauce-action-classification-metatype

An OWL Reasoner *(DNEG)*
https://github.com/sauce-consortia/sauce-action-owlreasoner

An Alembic Rescaler *(DNEG)*
https://github.com/sauce-consortia/sauce-action-transformation-scalealembic

A Texture Classifier *(DRZ)*
https://github.com/sauce-consortia/sauce-action-classification-texturetag

A Turntable Blender Transformer *(DRZ)*
https://github.com/sauce-consortia/sauce-action-turntable-blender

An OBJ and Photo Classifier *(DRZ)*
https://github.com/sauce-consortia/sauce-action-classification-textags

An Alembic to USD Transformer *(DNEG)*
https://github.com/sauce-consortia/sauce-action-transformation-alembic2usd

A USD to Image Renderer *(Foundry)*
https://github.com/sauce-consortia/USD-renderer

In the subsequent sections we will cover how the various webhooks also feature and are leveraged in the execution framework.

7 The Execution Framework

In D4.1 Smart Search Framework POC, we introduced a framework for executing, composing and chaining together classifiers and transformers and we demonstrated this in D4.2, D5.2 and D5.3. This was based on Apache Openwhisk, a serverless framework supporting multiple runtimes, deployed on top of Kubernetes. Visual effects houses usually have a variety of runtimes and compute frameworks for rendering, transformation, compositing etc. So in order to extract the most value from contributions, a variety of components were developed to support a cross section of platforms and runtimes in order to execute classification, transformation, chaining and composition. These are detailed below:

7.1 OpenWhisk

In D4.1 we demonstrate a proof of concept for the smart search framework using OpenWhisk as the execution framework. A number of components, including baseactions and manifests, were provided to help VFX houses deploy, execute and chain classifiers and transformers explicitly. This was further extended to include a powerful command line interface and platform wrapper enabling the execution framework to be run and tested locally.

In addition to this, the entire framework was also tested and deployed on the compute platform (Kubernetes) and contributions were deployed and executed. The project below demonstrates this:

https://github.com/sauce-consortia/sauce-solution-ingestsearch

7.2 Kubernetes

As detailed in section 6.2, a new base image with various hooks and webhooks was developed in order to integrate with synchronous and asynchronous messaging systems. An archetypal boilerplate was provided that enables a classifier or transformer to be automatically deployed into kubernetes and exposed as service with a number of features and qualities detailed below.

Cache

A Python cache is available to all services to enable reuse of data and models across processes. When the container starts, it will run the setup() method in /action/setup.py only if the script exists. The container has a cache in /action/cache.py exposed as c, which can be imported and updated during setup or request execution. Example below:

```
from cache import c
```

```
import tensorflow
import numpy as np
from tensorflow import keras

def setup():
      c["model"] = keras.models.load_model('/model/model.h5')
      c["labels"] = np.load('/model/labels.npy')
```

Hooks

There are two hooks available, a required exec hook and an optional after hook. Hooks can either be mounted into containers using volumes or written into an image in a Dockerfile by extending the base image and specifying the correct environment variables.

The base image ships with an ‘exec’ hook built-in, which the script will fallback to if no exec hook is explicitly specified. This script will be executed for each request with a number of parameters passed through.

An environment variable should be set and point to where you have located your exec script. If this is not specified, it will default to app/hooks/exec, which is the built-in exec script.

Sync versus non blocking

By default, the image processes requests synchronously and provides a response when the entire sequence of events has completed (as described in the basic request response). The client can make an asynchronous request by specifying the Prefer header in the request as per RFC7240:

```
Prefer: respond-async
```

The container can then decide whether to behave asynchronously or synchronously.

Publishing via Webhooks

The base image ships with an optional, built-in publishing mechanism that can be configured using the PUBLISH_ENDPOINT environment variable. This allows you specify at deployment time where you would all classification results and errors to be published to. This is achieved via a POST request with the output of exec as the body of the request to the endpoint specified in PUBLISH_ENDPOINT.

You can also specify in the request a callback webhook with the X-Callback-URL header set to your webhook endpoint when you make a classification request.

```
X-Callback-URL: http://example.com/webhook
```

You will get a 202 response and a request header X-Request-Id specifying the request ID that will be in the X-Correlation-Id header of the resulting request against your webhook endpoint.

Publishing via Google Pub/Sub

The base image ships with an optional built-in publishing mechanism for Google Pub/Sub that can be configured using the GOOGLE_PUBSUB_PROJECT and GOOGLE_PUBSUB_TOPIC environment variables. This publishes the response entity to the project topic, and continues with any other processing or webhooks defined above.

Pub/Sub Operator and Custom Resource Definitions

In addition to providing hooks into Pub/Sub topics, the execution framework also provides a Pub/Sub operator and custom resource definitions that enables classifiers and transformers deployed into the framework to bootstrap topics for chaining and composition of multi step operations. The project below demonstrates this:

https://github.com/sauce-consortia/pubsub-operator

When a classifier or transformer is deployed into the framework using the boilerplate, a Kubernetes Operator provisions a new topic for the component to publish outputs of classification and transformation, allowing an orchestrator to collect them as inputs for classification and transformation chaining.

8 The Search Framework

The search framework is split into two distinct and decoupled components, independently deployable and usable. They are the search API, a service that allows an application to execute searches against asset data, and a search UI that provides a user experience and user interface for humans to interrogate, search, filter and navigate assets.

8.1 The Search API

The search framework provides a search API with a number of resources and features that enable interrogation of assets using a variety of means.

https://github.com/sauce-consortia/graphs-api

These endpoints and methods enable different applications and user experiences to leverage and access data in different ways, whilst conforming to a uniform and mature set of standards. The main resources and key features of the Search API are listed below:

SPARQL 1.1 API's

The Search API provides a fully conformant SPARQL1.1 endpoint with all the features associated with the standard. This supports SPARQL Protocol, SPARQL Update, SPARQL Graphstore and SPARQL Federation and allows full access to the assets and associated vocabularies through the default graph.

GraphQL API

The search API also provides an endpoint supporting a limited subset of GraphQL queries. This allows an application developers to interrogate the data using simpler syntax and provides data in data format closer to the asset model, at the cost of expressivity

Autosuggest API

An autosuggest API provides suggestions for tags, labels, synonyms and hypernyms using a standard HTTP GET operation on a URL, passing in a fragment of a word or phrase in order to get suggestions.

Asset API

There is also a fully fledged REST API for interacting with individual assets and depictions supporting a full range of HTTP operations. This provides a means to work with complete assets on an individual asset graph level in isolation.

Similarity API

One of the core capabilities of the Search Framework was to provide algorithms and methods for ascertaining similarity between assets. The search API includes a number of resources that enable similarity between individual and collective assets, which can be extended by implementing custom predicates to include more domain specific sub systems for predicting similarity.

Ingest API

There are also API's that allow for the ingest of one or more datasets both related and unrelated, supporting a variety of formats and processing methods. The Ingest API is the target of the outputs from the execution framework.

8.2 The Search User Interface

In D4.4 we demonstrated functionality for a search user interface for the Smart Search Prototype and have further tested and documented it's efficacy in the combined evaluation report in D8.5. This section details some of the technical and architectural aspects of the user interface.

Technical Overview

The user interface is developed and packaged as a ReactJS application that can be accessed on a URL from a modern browser. It also leverages a number of React Bootstrap components and uses a number of modern browser features such as local storage and caching to improve performance and user experience

https://github.com/sauce-consortia/sauce-dist-ui

One of the key aspects of the application is to also provide renders for different types of asset and many of these are enabled by integrating a number of three.js components. The project above details how additional renderers can be added and configured to the user interface

Architectural Overview

The user interface follows an Atomic Design Methodology, roughly propentising the use of atoms, molecules and organisms. Atoms represent primitive user interface controls that can be grouped together as molecules, coupled components with varying behaviours. Organisms represent complete interfaces, widgets, and applications, consisting of one or more molecules and themes.

By adopting this methodology, different components and parts of the user interface can be used, reused and deployed entirely independently, either as distinct applications, or part of existing or third party applications. This is incredibly valuable in Visual Effects Houses where

they employ a plethora of applications and interfaces across the production pipelines and being able to present search components and behaviours in different contexts provides greater productivity more effectively.

9 Further Development

Over the course of the project and during development, a number of opportunities for further development presented themselves. The project as whole provides a number of foundational and potential leads for further research and development from both academic and industry partners. Detailed below are a few of them.

9.1 Relationship to Universal Scene Description

The project as a whole made a number of assumptions about the role of Universal Scene Description and its suitability as a standard for asset descriptions and asset interchange. In 2.3, extensive research was carried out to assess the merits, applicability and usefulness of different scene description languages for a variety of use cases, including search. Whilst many scene description compliment search, had support for metadata and had mature and widely adopted asset models, they fell short of fully meeting the use cases and requirements listed in D2.1, and were designed more as interchanges between authoring and editing tools, rather than providing a model for asset discovery. To this effect a generalised asset descriptor for search was proposed and developed, based on existing data standards.

That said, over the course of the project what transpired was that it became increasingly apparent that there was a strong case for a standard scene description for the purpose of *classification* and by providing a means of homogenising any number of proprietary formats into something standard and widely accepted, made classification more effective.

Much of D4.4 describes different means of classification and the role of turntable and 2d rendering and a large number of contributions focussed on not training classification models on actual asset data (e.g OBJ, USD, Alembic etc.), rather producing rendering and turntable algorithms from a standard scene description, and classifying the resultant outputs. This certainly aligns with DNEG's strategic shift towards Universal Scene Description and most likely aligns with existing industry trends.

9.2 European Union Linked Open Vocabularies

Much of the project focussed on developing and sharing vocabularies and ontologies covering different VFX domains. The European Union maintains a number of Linked Open Vocabularies for use and reuse across a variety of domains.

https://joinup.ec.europa.eu

The contributions created as a result of this project, along with any future contributors, would be invaluable to consumers and users of linked open vocabularies, and various intersections and extensions of other open vocabularies may provide insightful and should be considered as contributions.

9.3 W3C Community and Business Groups

There are a number of W3C Community and Business Groups involved in Media, Production and Film. The SAUCE project and associated outputs would be welcome additions to any one of these existing groups, or alternatively provide the foundation for a new group.

https://www.w3.org/community/groups/

The vocabularies and design of the outputs leverages W3C standards across the board, so proposals and requests would be received very favourably. In addition it would offer the opportunity to collaborate with other industry and academic partners.

9.4 Open Sourcing and Third Party Integration

Throughout the course of the project a number of opportunities for dissemination and collaboration with external agencies has been evident and effected. The dissemination plan details and summarises the majority of these. However from a purely ground up level, opportunities for additional forms of technical integration with existing open source communities are also apparent. Whilst legal and commercial agreements officiate many of these, it's worth mentioning some of the potential use cases.

A number of operators for Kubernetes were developed, and these could be shared with existing operator contribution sites. There are a number of active communities sharing and supporting these and the components developed during SAUCE could be a good fit.

Finally, we discussed using Flix as storage in section 5 of this document. We described how many aspects of the platform were provided using Terraform. Terraform provides a declarative way for provisioning and updating infrastructure. It also supports a rich and active ecosystem for third party extensions and plugins. Extending this with Flix and SAUCE components would be one useful way to proliferate the frameworks and to provide cleaner and more effective setups and installations.

10 Conclusion

This document provides a comprehensive overview of the Smart Search Framework and associated parts. The SAUCE project was developed over the course of three years and this document details and describes the most salient part of work packages in collaboration, emphasising contribution and integration. The architecture and components were designed to meet the immediate use cases of the project, but also to provide a means for further research and development. By decoupling components into sub-systems and smaller frameworks, we have provided a better means for doing this and allowing users and contributors to pick and choose how they interact with various components. This, combined with the adherence to existing standards along with best practice in the development lifecycle, means that industry and academia can more easily and adopt the concepts and outputs of the project.

11 Web references

Prefer RFC
https://tools.ietf.org/html/rfc7240

OpenWhisk - A serverless framework
https://openwhisk.apache.org/

Google Cloud - cloud provider
https://cloud.google.com

Universal Scene Description - open scene description language from Pixar
https://graphics.pixar.com

Open Image IO - open source library for working with a variety of image formats
https://github.com/OpenImageIO/oiio

Terraform - declarative cloud agnostic provisioning and configuration management tool
https://www.terraform.io/

W3C Community Groups
https://www.w3.org/community/groups/

Linked Open Vocabularies
https://joinup.ec.europa.eu

Atomic Design Methodology
https://atomicdesign.bradfrost.com/

Stemming and Lemmatization
https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

SPARQL - Sparql and RDF Query Language
https://www.w3.org/TR/sparql11-query/

ReactJs - open source javascript framework for building user interfaces
https://reactjs.org/

ReactBootstrap
https://react-bootstrap.github.io/

ThreeJS - 3d javascript library and WebGL framework
https://threejs.org/

GraphQL
https://graphql.org/

NodeJs - javascript application framework
https://nodejs.org/en/

Typescript - javascript programming dialect
https://www.typescriptlang.org/

Kubernetes - container orchestration and management system
https://kubernetes.io/

RDF
https://www.w3.org/standards/techs/rdf

RDFS
https://www.w3.org/TR/2014/REC-rdf-schema-20140225/

OWL2
https://www.w3.org/TR/owl2-overview/

GitOps - methodology for cloud and configuration management
https://www.weave.works/technologies/gitops/