A set of Common Service Quality Assurance Baseline Criteria for Research Projects

A DOI-citable version of this manuscript is available at https://doi.org/10.20350/digitalCSIC/12533.

This manuscript was automatically generated on 29-04-2020.

Authors

- **Pablo Orviz**
  0000-0002-2473-6405 · orviz
  Spanish National Research Council (CSIC); Institute of Physics of Cantabria (IFCA)

- **Mario David**
  0000-0003-1802-5356 · mariojmdavid
  Laboratory of Instrumentation and Experimental Particle Physics (LIP)

- **Jorge Gomes**
  0000-0002-9142-2596 · jorge-lip
  Laboratory of Instrumentation and Experimental Particle Physics (LIP)

- **Joao Pina**
  0000-0001-8959-5044 · jopina
  Laboratory of Instrumentation and Experimental Particle Physics (LIP)

- **Samuel Bernardo**
  0000-0002-6175-4012 · samuelbernardolip
  Laboratory of Instrumentation and Experimental Particle Physics (LIP)

- **Isabel Campos**
  0000-0002-9350-0383 · isabel-campos-plasencia
  Spanish National Research Council (CSIC); Institute of Physics of Cantabria (IFCA)

- **Germán Moltó**
  0000-0002-8049-253X · gmolto
  Universitat Politècnica de València (UPV)

- **Miguel Caballer**
  0000-0001-9393-3077 · micafer
  Universitat Politècnica de València (UPV)

- **Vyacheslav Tykhonov**
  0000-0001-9447-9830 · 4tikhonov
  DANS-KNAW

Abstract

The purpose of this document is to define a set of quality standards, procedures and best practices to conform a Service Quality Assurance plan, to serve as a reference within the European research ecosystem related projects for the adequate development, deployment, operation and integration of services into production research infrastructures.

Copyright Notice

Acknowledgements

The EOSC-Synergy project received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement number 857647.

Document Log

Issue	Date	Comment
v0.1	27/04/2020	First draft version
v0.2	28/02/2020	Second draft version
v1.0-beta03	09/06/2020	beta03 draft version
v1.0	12/06/2020	v1.0 release
v2.0	02/02/2022	Issues: #6, #15, #23, #25, #27, #28, #29, #30, #31, #32, #33, #34, #35, #36
v2.1	09/11/2022	Issues: #43, #47, #49, #50

1. Introduction

The Open Science realization in Europe is implementing the European Open Science Cloud (EOSC). The EOSC aims at providing researchers with a unique, federated and inclusive view of fit-for-purpose services, developed and operated by the diverse European research infrastructures, including the underlying e-Infrastructures. Consequently, the ultimate success of the EOSC heavily relies on the quality aspects of those services, such as their stability or functional suitability.

The meaning of **Service** can be regarded from different perspectives. From an IT Service Management (ITSM) standpoint, such as the EOSC Service Management System (SMS) process model, a **Service** is devised as a means to "provide value to the customer". The same goal is shared by the DevOps paradigm, but in this case there is a more pragmatic vision that the customer satisfaction is achieved through the continuous delivery of quality-assured **Services**, with a shorter life cycle, as the final outcome of a comprehensive software development process.

The ITSM model has a broader focus. A **Service** is an "intangible asset" that also includes additional activities such as customer engagement and support. Consequently, it is a much heavier process that might not be appropriately applicable for all types of **Services**. The DevOps model, on the other hand, narrows down the scope to meet the user expectations by acting exclusively on the quality features of the **Service**, which is seen as an aggregate of software components in operation.

2. Purpose

This document provides an initial approach to **Service Quality Assurance**, meant to be applied in the integration process of the **Services** existing under the EOSC-Synergy project, which eventually will be accessible as part of the EOSC offerings.

The criteria compiled in this document favours a pragmatic and systematic approach, putting emphasis on the programmatic assessment of the quality conventions. As such, the criteria herein compiled builds on the DevOps culture already established in the Software Quality Assurance baseline document [1], to outline the set of good practices that seek the usability and reliability of **Services**, and meet the user expectations in terms of functional requirements.

3. Contextualization of a Service

As a result, a **Service**, as conceived in this document, represents the following:

- **Web Service** [2]:
  - A **Web Service** is an application or data source that is accessible via a standard web protocol (HTTP or HTTPS).
  - **Web Services** are designed to communicate with other programs, rather than directly with users.
  - Most **Web Services** provide an API, or a set of functions and commands, that can be used to access the data.
- **Web Application** [3]:

  - A **Web Application** or “Web App” is a software program that is delivered over the Internet and is accessed through a web browser.

- **Platform / Service Composition** [4]:

  - Aggregation of multiple small services into larger services, according to a service-oriented (SOA) and/or microservices architecture.

  - An integrated set of **Web Services**, **Web Applications** and software components.

Examples are: Web portals, Scientific portals and gateways, data repositories.

4. Goals

The herein proposed baseline, harnesses the capabilities of the quality factors in the underlying software to lay out the principles for attaining quality in the enabled services within the EOSC context. According to this view, service quality is the foundation to shape user-centric, reliable and fit-for-purpose services.

The Service Quality baseline aims at fulfilling the following goals:

1. Complement with a DevOps approach the existing approaches to assess and assure the quality and maturity of services within the EOSC, i.e. Technology Readiness Levels (TRLs) and EOSC Service Management System (SMS).

2. Build trust on the users by strengthening the reliability and stability of the services, with a focus on the underlying software, thus ensuring a proper realization of the verification and validation processes.

3. Ensure the functional suitability of the service by promoting testing techniques that check the compliance with the user requirements.

4. Improve the usability by identifying the set of criteria that fosters the service adoption.

5. Promote the automated validation of the service quality criteria.

5. Notational Conventions

The keywords “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119 [5].

6. Quality Criteria - Automation

The following sections describe the quality conventions and best practices that apply to the development, operation and integration phases of a **Service** with a production infrastructure for research, such as the EOSC ecosystem. These guidelines rule the **Service** development and operation process within the framework of the EOSC-Synergy project.

The criteria in this document complements the criteria described in the “Software Quality Assurance baseline” [1], while following the same pragmatic DevOps approach of automation.

6.1. Deployment

6.1.1. Automated Deployment [SvcQC.Dep]

The automated deployment of **Services** implies the use of code to install and configure them in the target infrastructures. Infrastructure as Code (IaC) templates allow operations teams to treat service provisioning and deployment in a similar fashion as developers manage the software code.

Consequently, IaC enables the paradigm of immutable infrastructure deployment and maintenance, where **Services** are never updated, but deprovisioned and redeployed. An immutable infrastructure simplifies maintenance and enhances repeatability and reliability.

- **[SvcQC.Dep01]** A production-ready **Service SHOULD** be deployed as a workable system with the minimal user or system administrator interaction leveraging IaC templates.

- **[SvcQC.Dep02]** Any future change to a deployed **Service SHOULD** be done in the form of a new deployment, in order to preserve immutable infrastructures.

- **[SvcQC.Dep03]** IaC **SHOULD** be validated by specific (unit) testing frameworks for every change being done.

    - **[SvcQC.Dep03.1]** IaC (unit) tests **MUST** be idempotent.

6.2. Dynamic testing - Black box testing

6.2.1. API Testing [SvcQC.Api]

Web services commonly use application programming interfaces (APIs) to expose the available features to external consumers, which can be either oriented to the end-user or suitable for machine-to-machine communications.

Accurate implementation of a publicly-accessible API, is driven by a clearly defined specification. The OpenAPI Specification (OAS) [6] provides the most suitable way to describe, compose, consume and validate APIs. The following requirements assume the presence of such an API specification.

- **[SvcQC.Api01]** API testing **MUST** cover the validation of the features outlined in the specification (aka *contract testing*).

    - **[SvcQC.Api01.1]** Any change in the API not compliant with the OAS **MUST NOT** pass contract testing.

    - **[SvcQC.Api01.2]** The use of OAS **SHOULD** narrow down the applicable set of test cases to the features described in the specification, avoiding unnecessary assertions.

- **[SvcQC.Api02]** API testing **MUST** include the assessment of the security-related criteria outlined in SvcQC.Sec section.

- **[SvcQC.Api03]** API testing **SHOULD** involve the use of test doubles, such as mock servers or stubs, that act as a validation layer for the incoming requests.

6.2.2. Integration Testing [SvcQC.Int]

Integration testing refers to the evaluation of the interactions among coupled **Services** or parts of a system that cooperate to achieve a given functionality.

- **[SvcQC.Int01]** Whenever a new functionality is involved, integration testing **MUST** guarantee the operation of any previously-working interaction with external **Services**.

    - **[SvcQC.Int01.1]** When using APIs, contract testing **MUST** detect any disruption in the communication between provider and consumer endpoints, through the validation of the API specification SvcQC.Api01.

- **[SvcQC.Int02]** Integration testing **MUST NOT** rely on non-operational or out-of-the-warranty services.

- **[SvcQC.Int03]** On lack of automation, ad-hoc pilot **Service** infrastructures and/or local testbeds **MAY** be used to cope with the integration testing requirements.

- **[SvcQC.Int04]** In the presence of CI environments, integration tests **SHOULD** be suitable for the automated testing.

6.2.3. Functional tests [SvcQC.Fun]

Functional testing is a type of black-box testing. It involves the verification of the **Service** identified functionality, based on requested requirements and agreed design specifications. This type of **Service** testing focuses on the evaluation of the functionality that the **Service** exposes, leaving apart any internal design analysis or side-effects to external systems.

- **[SvcQC.Fun01]** Functional testing **SHOULD** tend to cover the full scope –e.g. positive, negative, edge cases– for the set of functionality that the **Service** claims to provide.

    - **[SvcQC.Fun01.1]** When using APIs, contract testing **MUST** detect any disruption in the features exposed by the provider to the consumer, through the validation of the API specification. SvcQC.Api01.

    - **[SvcQC.Fun01.2]** Functional tests **SHOULD** include the Web Interface or Graphical User Interface (GUI) of the **Service**.

- **[SvcQC.Fun02]** Functional tests **SHOULD** be checked automatically.

- **[SvcQC.Fun03]** Functional tests **SHOULD** be provided by the developers of the underlying software.

6.2.4. Performance tests [SvcQC.Per]

Performance testing verifies that the software meets the specified performance requirements and assesses performance characteristics - for instance, capacity and response time [7].

Stress or Load testing, exercises software at the maximum design load, as well as beyond it, with the goal of determining the behavioral limits, and to test defense mechanisms in critical systems [7]. *Stress testing is a subset of Performance testing* [8].

Scalability testing is a test methodology in which an application’s or **Service** performance is measured in terms of its ability to scale up *and/or* scale out the number of user requests or other such performance measure attributes, through an increase in the amount of available resources. The definition is based on [9]. *Scalability testing is a subset of Performance testing*.

Elasticity is based on how quickly **Services** in an infrastructure are able to adapt [9], in response to variable demand or workload for those service(s) [10]. *Elasticity testing is a subset of Performance testing*.

- **[SvcQC.Per01]** Performance testing **SHOULD** be carried out to check the **Service** performance under varying loads.

- **[SvcQC.Per02]** Stress testing **SHOULD** be carried out to check the **Service** to determine the behavioral limits under sudden increased load.

- **[SvcQC.Per03]** Scalability testing **MAY** be carried out to check the **Service** ability to scale up or scale down when its load reaches the limits.

- **[SvcQC.Per04]** Elasticity testing **MAY** be carried out to check the **Service** ability to scale out or scale in, depending on its demand or workload.

6.2.5. Security [SvcQC.Sec]

Security assessment is essential for any production **Service.** While an effective implementation of the security requirements applies to every stage in the software development life cycle (SDLC) –especially effective at the source code level, as discussed in [1], section *Security **[SQA-QC.Sec]***–, the security testing of a **Service** is also –similarly to the diverse testing strategies previously covered– a black-box type of testing. Hence, this section focuses on the runtime analysis of security-related requirements, as part of the *Dynamic Application Security Testing* (*DAST*) as well as the *Interactive Application Security Testing* (*IAST*).

Additionally, the compliance with security policies and regulations complements the analysis, which can be implemented, continuously validated and monitored through the *Security as Code* (*SaC*) capabilities. SaC is a particularly suitable tool for endorsing security of **Service Composition** deployments.

- **[SvcQC.Sec01]** The **Service** public endpoints and APIs **MUST** be secured with data encryption.

  - **[SvcQC.Sec01.1]** The **Service MUST** use strong ciphers for data encryption.

- **[SvcQC.Sec02]** The **Service SHOULD** have an authentication mechanism.
    - **[SvcQC.Sec02.1]** Whenever dealing with a **Service Composition**, such as microservice architectures, the **Services SHOULD** be managed by a centralized authentication mechanism.
    - **[SvcQC.Sec02.2]** In publicly-accessible APIs, **Service** authentication **SHOULD** be handled through an API gateway in order to control the traffic and protect the backend services from overuse.
- **[SvcQC.Sec03]** The **Service SHOULD** implement an authorization mechanism.
    - **[SvcQC.Sec03.1]** In **Service Composition** environments, the authorization mechanism **SHOULD** uniquely grant the essential access permissions for each **Service** according to the *Principle of Least Privilege* (*PoLP*).
- **[SvcQC.Sec04]** The **Service MUST** validate the credentials and signatures.
    - **[SvcQC.Sec04.1]** Credentials used in the **Service MUST** be signed by a recognized and trusted certification authority.
- **[SvcQC.Sec05]** The **Service MUST** handle personal data in compliance with the applicable regulations, such as the General Data Protection Regulation (GDPR) within the European boundaries.
- **[SvcQC.Sec06]** The **Service SHOULD** be audited in accordance with the black-box testing criteria identified by de-facto (cyber)security standards and good practices.
    - **[SvcQC.Sec06.1]** Dynamic application security testing (DAST) checks **MUST** be executed, through the use of ad-hoc tools, directly to an operational **Service** in order to uncover runtime security vulnerabilities and any other environment-related issues (e.g. SQL injection, cross-site scripting or DDOS). The latest release of OWASP's Web Security Testing Guide [11] and the NIST's Technical Guide to Information Security Testing and Assessment [12] **MUST** be considered for carrying out comprehensive **Service** security testing.
    - **[SvcQC.Sec06.2]** Interactive Application Security Testing (IAST) [13], analyzes code for security vulnerabilities while the app is run by an automated test. IAST **SHOULD** be performed to a service in an operating state.
    - **[SvcQC.Sec06.3]** Penetration testing (manual or automated) **MAY** be part of the application security verification effort.
    - **[SvcQC.Sec06.4]** The security assessment of the target system configuration is particularly important to reduce the risk of security attacks. The benchmarks delivered by the Center for Internet Security (CIS) [14] and the NIST's Security Assurance Requirements for Linux Application Container Deployments [15] **MUST** be considered for this task.
- **[SvcQC.Sec07]** IaC testing, from SvcQC.Dep02 criterion, **MUST** cover the security auditing of the IaC templates (*SaC*) in order to assure the deployment of secured **Services**. For all the third-party dependencies used in the IaC templates (including all kind of software artifacts, such as Linux packages or container-based images):
    - **[SvcQC.Sec07.1]** SaC **MUST** perform vulnerability scanning of the artefact versions in use.

- **[SvcQC.Sec07.2]** SaC **SHOULD** verify that the artifacts are trusted and digitally signed.

- **[SvcQC.Sec07.3]** SaC **MUST** scan IaC templates to uncover misalignments with widely accepted security policies from SvcQC.Sec06 criteria, such as non-encrypted secrets or disabled audit logs.

- **[SvcQC.Sec07.4]** SaC **MAY** be used to seek, in the IaC templates, for violations of security requirements outlined in the applicable regulations from criterion SvcQC.Sec05.

- **[SvcQC.Sec07.5]** World-writable files **SHOULD NOT** be created while the service is in operation. Whenever they are required, the relevant files **MUST** be documented.

- **[SvcQC.Sec07.6]** World-readable files **MUST NOT** contain passwords.

7. Quality Criteria - Operational

This section describes the operational quality criteria that is not fit for automation, but that contribute to the assessment of the quality of the service when it’s in an operational production state.

7.1. Files and documents

7.1.1. Documentation [SvcQC.Doc]

Documentation is an integral part of any Software or Service. For example, it describes how and what users can use and interact with it, or how operators can deploy, configure and manage a given Software or Service.

- **[SvcQC.Doc01]** Documentation **MUST** be available online, easily findable and accessible.
- **[SvcQC.Doc02]** Documentation **SHOULD** have a Persistent Identifier (PID).
- **[SvcQC.Doc03]** Documentation **MUST** be version controlled.
- **[SvcQC.Doc04]** Documentation **MUST** be updated on new **Service** versions involving any change in the installation, configuration or behavior of the **Service**.
- **[SvcQC.Doc05]** Documentation **MUST** be updated whenever reported as inaccurate or unclear.
- **[SvcQC.Doc06]** Documentation **SHOULD** have a non-software license.
- **[SvcQC.Doc07]** Documentation **MUST** be produced according to the target audience, varying according to the **Service** specification. The identified types of documentation and their RECOMMENDED content are:
    - **[SvcQC.Doc07.1]** Deployment and Administration:
        - Installation and configuration guides.
        - Service Reference Card, with the following RECOMMENDED content:
            - Brief functional description.
            - List of processes or daemons.
            - Init scripts and options.
            - List of configuration files, location and example or template.
            - Log files location and other useful audit information.
            - List of ports.
            - Service state information.

        - List of cron jobs.
        - Security information.
        - FAQs and troubleshooting.
    - **[SvcQC.Doc07.2]** User:
        - Detailed User Guide for the **Service**.
        - Public API documentation (if applicable).
        - Command-line (CLI) reference (if applicable).

7.1.2. Policies [SvcQC.Pol]

Policy documents describe what are the user’s expected behavior when using the **Service**, how they can access it and what they can expect regarding privacy of their data.

- **[SvcQC.Pol01]** The **Service MUST** include the following policy documents:
    - **[SvcQC.Pol01.1]** Acceptable Usage Policy (AUP): Is a set of rules applied by the owner, creator or administrator of a network, **Service** or system, that restrict the ways in which the network, **Service** or system may be used and sets guidelines as to how it should be used. The AUP can also be referred to as: Acceptable Use Policy or Fair Use Policy.
    - **[SvcQC.Pol01.2]** Access Policy or Terms of Use: represent a binding legal contract between the users (and/or customers), and the Provider of the **Service**. The Access Policy mandates the users (and/or customers) access to and the use of the Provider’s **Service**.
    - **[SvcQC.Pol01.3]** Privacy Policy: Data privacy statement informing the users (and/or customers), about which personal data is collected and processed when they use and interact with the **Service**. It states which rights the users (and/or customers) have regarding the processing of their data.

7.2. Support

7.2.1. Support [SvcQC.Sup]

Support is the formal way by which users and operators of the **Service** communicate with other operators and/or developers of the **Service** in case of problems, be it operational problems or bugs in the **Service** or underlying Software. Reporting of enhancements, improvements and documentation issues.

- **[SvcQC.Sup01]** The **Service MUST** have a tracker or helpdesk for operational and users issues.
- **[SvcQC.Sup02]** The **Service SHOULD** have a tracker for the underlying software issues [1], section [SQA-QC.Man01].
- **[SvcQC.Sup03]** The **Service SHOULD** include an Operational Level Agreement (OLA) with the infrastructure where it is integrated.

- **[SvcQC.Sup04]** The **Service** MAY include Service Level Agreement (SLA) with user communities.

7.3. Monitoring and Metrics

7.3.1. Monitoring [SvcQC.Mon]

Monitoring is a periodic testing of the **Service**. It requires a monitoring service from where tests are executed or sent and results of those tests are shown. The tests can be the same, in part or in total of the Functional, Security and Infrastructure tests. The technology used for the monitoring is left to the developers of the underlying software to decide eventually with input from the infrastructure(s), where the **Service** is foreseen to be integrated.

- **[SvcQC.Mon01]** The **Service** in an operational production state **SHOULD** be monitored for functional-related criteria:
  - **[SvcQC.Mon01.1]** The **Service** public endpoints **MUST** be monitored.
  - **[SvcQC.Mon01.2]** The **Service** public APIs **MUST** be monitored. Use functional tests of criteria SvcQC.Fun01.1.
  - **[SvcQC.Mon01.3]** The **Service** Web interface **MAY** be monitored. Use functional tests of criteria SvcQC.Fun01.2.
- **[SvcQC.Mon02]** The **Service MUST** be monitored for security-related criteria:
  - **[SvcQC.Mon02.1]** The **Service MUST** be monitored for public endpoints and APIs secured and strong ciphers for encryption. Use Security tests of criteria SvcQC.Sec01.
  - **[SvcQC.Mon02.2]** The **Service SHOULD** be monitored with DAST checks. Use Security tests of criteria SvcQC.Sec06.1.
- **[SvcQC.Mon03]** The **Service MUST** be monitored for infrastructure-related criteria:
  - **[SvcQC.Mon03.1]** IaC (unit) tests SvcQC.Dep02 **SHOULD** be reused as monitoring tests, thus avoiding duplication.

7.3.2. Metrics [SvcQC.Met]

A metric is a quantifiable measure used to track and assess the status of a specific process.

In the case of **Services**, some relevant metrics are the number of users registered in the **Service**, or of active users. Also accounting is important to track resource usage per user or group of users, either or both computing and storage resources.

Although the metrics may be published in external services managed by the infrastructure, this is a common case in federated infrastructures such as EOSC.

- **[SvcQC.Met01]** The **Service SHOULD** implement the collection of metrics.
  - **[SvcQC.Met01.1]** The collection of metrics **SHOULD** be cumulative over time and timestamped, so that the values can be queried per time interval.

- **[SvcQC.Met01.2]** The metric *Number of registered users* **SHOULD** be collected.

- **[SvcQC.Met01.3]** The metric *Number of active users over a given period of time* **MAY** be collected.

- **[SvcQC.Met01.4]** The metric *Amount of computing resources per user or per group* **MAY** be collected. The metric unit depends on the type of service and infrastructure. An example is `CPU x hours`.

- **[SvcQC.Met01.5]** The metric *Amount of storage resources per user or per group* **MAY** be collected. The metric unit depends on the type of service and infrastructure. An example is `TByte x month`.

7. Glossary

API
Application Programming Interface

AUP
Acceptable Usage Policy

CLI
Command Line Interface

DAST
Dynamic Application Security Testing

EOSC
European Open Science Cloud

GDPR
General Data Protection Regulation

GUI
Graphical User Interface

IaC
Infrastructure as Code

IAST
Interactive Application Security Testing

ITSM
IT Service Management

NIST
National Institute of Standards and Technology

OAS
OpenAPI Specification

OLA
Operational Level Agreement

OWASP
Open Web Application Security Project

PID
Persistent Identifier

PoLP
Principle of Least Privilege

SaC
Security as Code

SDLC
Software Development Life Cycle

SLA
Service Level Agreement

SMS
Service Management System

SOA
Service Oriented Architecture

VCS
Version Control System

A1. Annex

The Quality Criteria described in this document follows a technology-agnostic approach. As such the choice of tools and services to implement the workflow for service quality assessment process, is up to the team or community developing or operating a given service.

This annex describes an implementation approach to help service developers and operators, cover the Quality Criteria detailed in this document.

A1.1. Code workflow

The workflow for service quality assessment, is shown in Figure 1. It depicts a real case example, GitHub is used in particular to host the Dockerfiles to build the service Docker image, it also hosts the Ansible role for the service deployment and configuration, thus the IaC (Infrastructure as Code).

The SQA as a Service (SQAaaS), is used to create the Jenkinsfile (pipeline) to be executed in the Jenkins service.

Before the Jenkins pipeline execution phase, the figure depicts the build and upload/publish of the service Docker image in the Docker Hub.

The Jenkins service is the main service used to execute he pipeline. The figure show a simple pipeline with automated deployment of the service, execution of functional tests and dynamic security tests, but in general the pipeline can be more complex and execute other tools to assess the other service quality criteria.

The bottom part of the figure shows the services, tools and infrastructures that are actuated by the Jenkins pipeline execution: The IM (Infrastructure Manager) is the service that is used to deploy the service in cloud resources, fetching the corresponding Ansible role from Ansible Galaxy to instantiate and configure it.

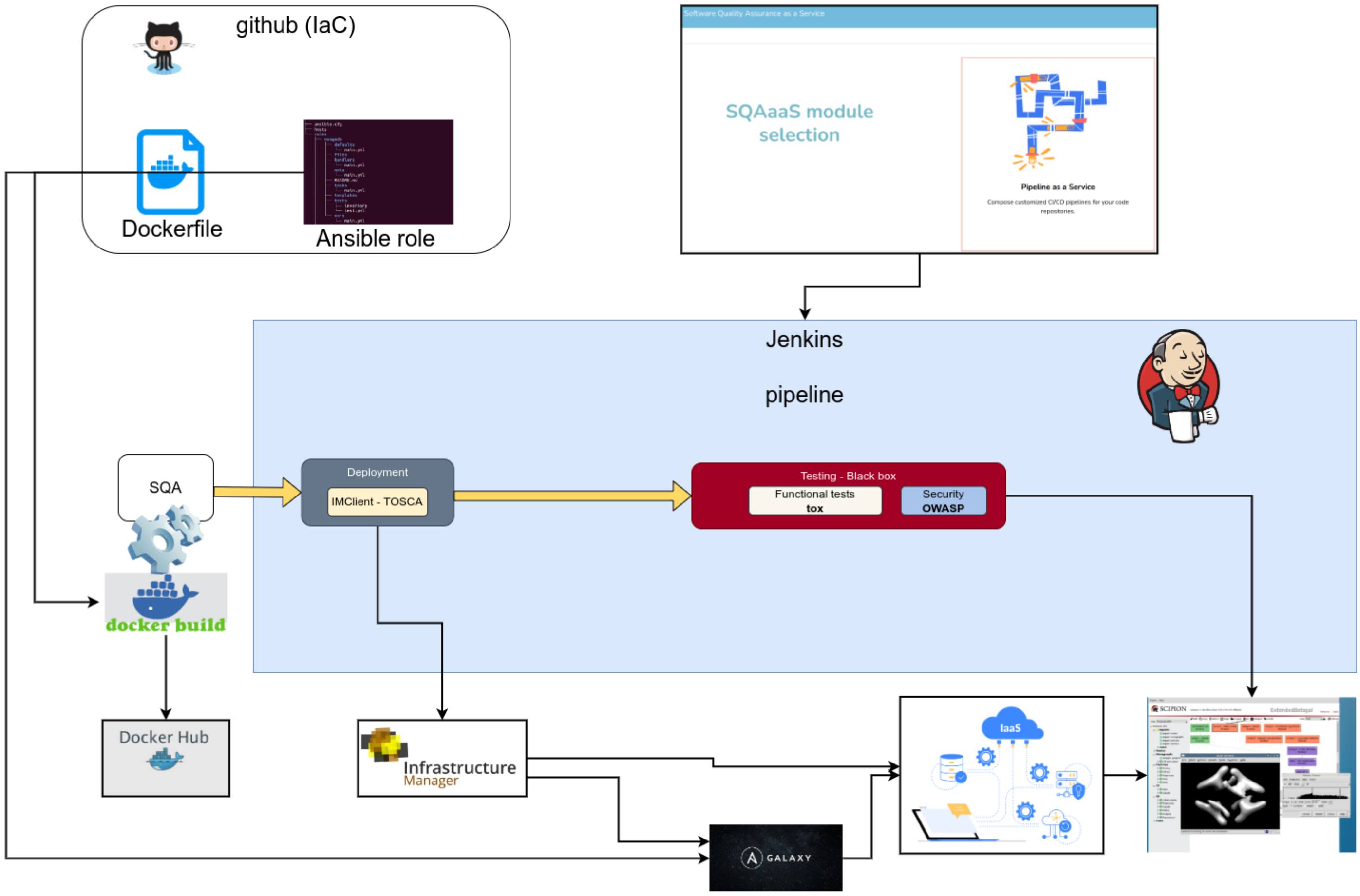

Figure 1: Workflow for service quality assessment

A1.2. Services

As it can be seen in Figure 1, multiple services take part in the execution of the CI/CD pipelines. Table 1 shows the list of services used for the service for the Quality Criteria assessment detailed in this document.

One of the most popular services for Software source code management is GitHub. It is uses Git as the Version Control System, branching and tag management.

Ansible and Ansible-Galaxy are used in many software frameworks as deployment and configuration tools.

The software is packed/built into executable artifacts that can be RPMs (case of RedHat and derivative OS), DEBs (case of Debian/Ubuntu and derivatives) and in many cases containers such as Docker images.

The artifacts are provided, in general, by public repositories and most notably Docker Hub [16/] in the case of Docker images.

Regarding the CI/CD automation, Jenkins pipelines can be easily composed through the SQAaaS platform and put into the git repositories to be used by the Jenkins CI service to perform the tests. The tools used in the CI automation are shown in section A1.2.

Kubernetes is a container management platform where services or platforms can be deployed while the IM (Infrastructure Manager) can also be used to automatically deploy services both in cloud resources or kubernetes clusters.

Service	Usage	Criteria	Repo URL or documentation	Comment
GitHub	VCS	**SvcQC.Dep**	https://docs.github.com/	Source code repository - git
Ansible, Galaxy	Install, Config	**SvcQC.Dep**	https://docs.ansible.com/	Automated deployment and configuration
SQAaaS platform	Pipeline composition	**All**	https://sqaaas.eosc-synergy.eu	Pipeline composition for automatic tests
SQAaaS platform	Assessment & awarding	**All**	https://sqaaas.eosc-synergy.eu	Badge awarding
Jenkins CI service	Automated tests	**All**	https://www.jenkins.io/	Execution of automatic tests
Docker Hub	Docker images	N.A.	https://hub.docker.com/	Public repository of Docker images
Kubernetes	Service deployment/management	**SvcQC.Dep**	https://kubernetes.io/	Docker container management
Infrastructure Manager (IM)	Service deployment	**SvcQC.Dep**	https://www.grycap.upv.es/im	Service deployment

Table 1: Tools and services used to implement the Service QA criteria, also shown the criteria where applicable.

A1.3. Tools for CI/CD

This section shows the tools being used in the CI pipelines and the criteria that it verifies. This list is based on the template file in https://github.com/EOSC-synergy/sqa-composer-templates/blob/main/tooling.json.

Tool	Criteria	Language	Repo URL or documentation	Summary
kubectl	**SvcQC.Dep**	Agnostic	https://kubernetes.io/docs/tasks/tools/	Automated deployment
im_client	**SvcQC.Dep**	RADL/TOSCA	https://imdocs.readthedocs.io/en/latest/client.html	Automated deployment
ec3_client	**SvcQC.Dep**	RADL/TOSCA	https://github.com/grycap/ec3	Automated deployment
tox	**SvcQC.Fun**	Python	https://tox.readthedocs.io/	Automated test framework

References

1. **A set of common software quality assurance baseline criteria for research projects**
Pablo Orviz, Álvaro López García, Doina Cristina Duma, Giacinto Donvito, Mario David, Jorge Gomes
(2017) https://digital.csic.es/handle/10261/160086

2. **Web Service Definition** https://techterms.com/definition/web_service

3. **Web Application Definition** https://techterms.com/definition/web_application

4. **Service Composition - Glossary | CSRC**
CSRC Content Editor
https://csrc.nist.gov/glossary/term/Service_Composition

5. **Key words for use in RFCs to Indicate Requirement Levels**
Scott O Bradner
Internet Engineering Task Force (1997-03) https://datatracker.ietf.org/doc/rfc2119/

6. **Home**
OpenAPI Initiative
https://www.openapis.org/

7. **Guide to the Software Engineering Body of Knowledge, Version 3.0**
P Bourque, RE Fairley
(2014) http://www.swebok.org

8. **Difference between Performance and Stress Testing**
GeeksforGeeks
(2019-05-15) https://www.geeksforgeeks.org/difference-between-performance-and-stress-testing/

9. **Scalability, Elasticity, and Efficiency in Cloud Computing: a Systematic Literature Review of Definitions and Metrics**
Sebastian Lehrig, Hendrik Eikerling, Steffen Becker
Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures (2015-05-04) https://doi.org/gghzfq
DOI: 10.1145/2737182.2737185 · ISBN: 9781450334709

10. **Scalability analysis comparisons of cloud-based software services**
Amro Al-Said Ahmad, Peter Andras
Journal of Cloud Computing (2019-12) https://doi.org/ggtz94
DOI: 10.1186/s13677-019-0134-y

11. **WSTG - Stable | OWASP Foundation** https://owasp.org/www-project-web-security-testing-guide/stable/

12. **Technical guide to information security testing and assessment.**
KA Scarfone, MP Souppaya, A Cody, AD Orebaugh
National Institute of Standards and Technology (2008) https://doi.org/gnkq9h
DOI: 10.6028/nist.sp.800-115

13. **What is IAST? Interactive Application Security Testing**
Veracode

https://www.veracode.com/security/interactive-application-security-testing-iast

14. **CIS Benchmarks™**
CIS
https://www.cisecurity.org/cis-benchmarks/

15. **Security assurance requirements for linux application container deployments**
Ramaswamy Chandramouli
National Institute of Standards and Technology (2017-10-11) https://doi.org/gnkq9g
DOI: 10.6028/nist.ir.8176

16. **Docker** https://hub.docker.com/