A Management Framework for Consolidated Big Data and HPC

Project number: 45924

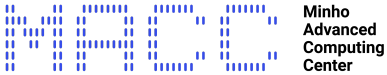
Deliverable 5.1 - Plan for software integration, validation and pilot activities

Date	31 March 2021
Activity	5 - Integration, Experimental Validation and Pilot
Version	v0.1
Disclosure	Public

Authors: Samuel Bernardo (LIP), Mário David (LIP)

Reviewers: João Paulo (INESC TEC), Bruno Antunes (Wavecom)

Partners



Funding

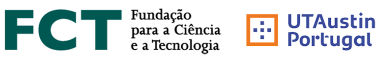


Table of Content

Executive Summary

In computing systems, the growing demand for High-Performance Computing (HPC) along with cloud infrastructures to answer the big data phenomenon, has been rising both in size and sophistication bringing another challenge for software development. Immersing more deeply in the topic, there are multiple aspects when reviewing the players in this ecosystem [FQJ+16]:

- Hardware and networking.
- Services software and platforms.
- Specialized applications or algorithms.
- Business models and education.

To understand this ecosystem, the BigHPC project provides a testbed to deploy the developed software components in an experimental environment, providing a collaborative environment to support the work between all involved teams. This environment takes into account all components from the BigHPC platform as well as the use cases that will assist in the validation.

TACC and MACC datacenters will provide the required resources to advance the development and deployment of the Pilot testbeds. These testbeds are integrated in the platform architecture and answer the demands from users and system administrators.

Glossary

API	Application Programming Interface
CI/CD	Continuous Integration/Continuous Delivery
CLI	Command Line Interface
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
GPU	Graphics Processing Unit
HPC	High-Performance Computing
I/O	Input/Output
INESC TEC	Institute for Systems and Computer Engineering, Technology and Science
LIP	Laboratory of Instrumentation and Experimental Particle Physics
MACC	Minho Advanced Computing Center
MPI	Message Passing Interface
RAM	Random-Access Memory
RDMA	Remote Direct Memory Access
REST	Representational State Transfer
SCM	Software Configuration Management
SDS	Software-Defined Storage
SQA	Software Quality Assurance
TACC	Texas Advanced Computing Center

1. Introduction

The practice of science requires nowadays a wide range of HPC software. However the usability of the scientific software and its development associated with engineering practices brings new challenges. Science requires a sustained, robust and reliable software ecosystem to answer the ever-growing demands for large scale simulation and data analysis [MR+19].

The goal of this deliverable is to provide the integration of the different software components developed within the project according to the BigHPC global platform architecture. A pilot testbed will be delivered enabling the integration and experimental validation of the several services and components under realistic conditions.

As such, the Pilot testbed infrastructures serves three main actors, the developers of components that have to test and validate their Software and perform the integration with other components, the system administrators of the platform in order to test the management of the several components and the users that want to test the platform as a whole by executing their applications and benchmarks.

The software components discussed in this document are based on the architecture and interfaces presented in Deliverable 1.2. The details are described in the following sections:

- Section 2: BigHPC platform modules overview and introduces the verification and validation process.
- Section 3: reviews the interfaces and the required testing environment.
- Section 4: describes the testbed for BigHPC platform.
- Section 5: closes this deliverable providing the conclusion.

2. Platform Verification and Validation

The set of components that make the BigHPC platform are divided into two major categories:

1. High-level platform components: namely the following components or services; the Orchestrator (composed by the Virtualization and Storage Managers), the repository of container images, the user and system administrator interfaces (composed by the Deployment, Management and Monitoring interfaces) and the Monitoring Backend (c.f. Deliverable 1.2).
2. Low-level software components that are installed on HPC cluster compute nodes, such as the monitoring probes or agents, that collect metrics and the SDS data plane stages that enforce rules sent by the Control Plane (Storage Manager).

The BigHPC platform can be seen as a collection of interlinked modules (services) that work and interact with each other. Therefore, the Pilot testbed infrastructure has to satisfy the needs of the developers regarding test and integration of all services and components of BigHPC, in order to achieve validation of the platform. The Pilot testbed serves both as a preview for end-users that can execute real-world applications and as a demonstrator for the system administrators that wish to test the management capabilities of the platform. Figure 1 depicts the high-level view of the Pilot testbed as well as the main actors of this infrastructure.

Since the platform has a modular architecture, each component or service can be tested and validated together with the remaining services, without impacting the other components. The validation can be performed once a new version of a given component becomes available. The modularization of components should allow an independent deployment of any given component including an upgrade or even downgrade or rollback in case of issues.

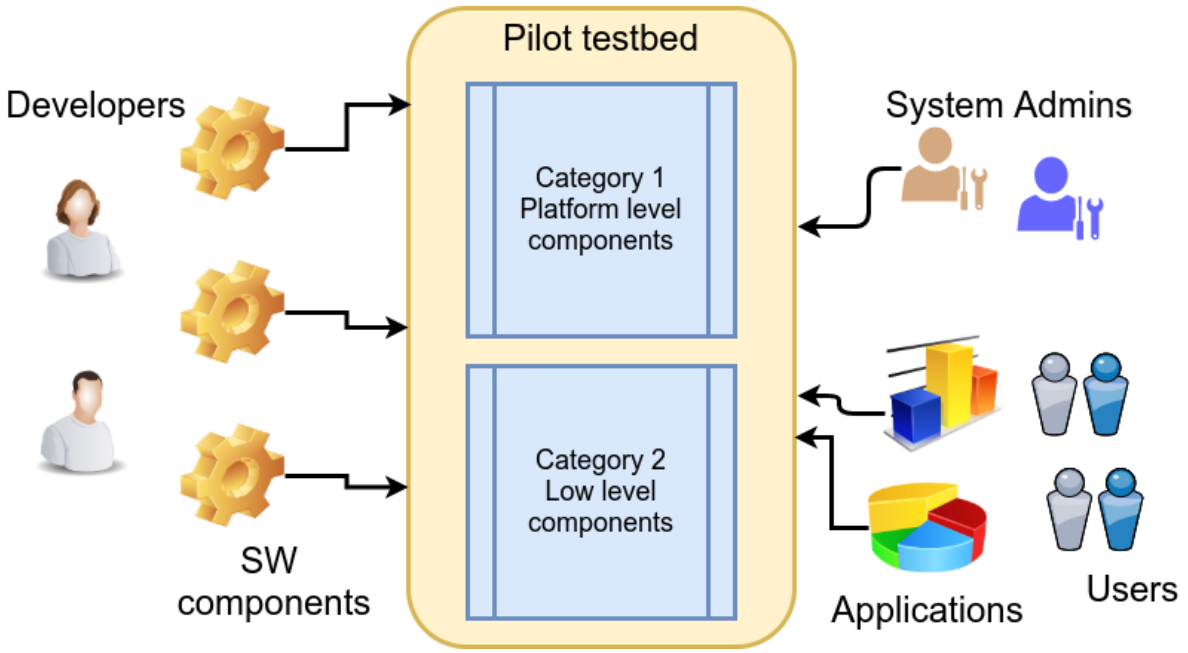


Figure 1: High-level overview of the Pilot testbed with the main actors of the infrastructure.

The components or services are further divided into stateless and stateful:

1. The stateless components can be deployed and configured in any infrastructure, without any dependency on pre-existing data or metadata. The testing of new versions of such components is simplified; the new version can be tested by upgrading to the new version, while in case of errors or bugs, the component can be quickly downgraded or replaced by the previous working version. Category 2 (low-level components) are all stateless.
2. In the case of stateful components, such as components that depend on a database, pre-existing data and/or metadata (c.f. the services for the container repositories), a copy should be performed prior to the upgrade of the component. In this way, if some error or bug occurs, and the upgrade implies a change on the database schema, then the component can be downgraded and the backup copy can be restored to a healthy state.

The integration, usability and performance evaluation activities start at the beginning of the project second year. These activities will culminate in the third year with the complete pilot deployment. The validation of the services and components that comprise the BigHPC platform will include a variety of practices that goes from hardware testing to the services

itself, ensuring performance, monitoring, expected operations or any other triage as expected.

Platform verification and validation is represented by a pipeline as code following DevOps practices such as Continuous Integration (CI) and Continuous Delivery (CD). This pipeline consists of a chain of processing elements arranged in stages aimed to investigate whether a software system fulfils the required purpose and satisfies specifications and standards.

Verification builds up the processes to check that a software achieves its goals without bugs and the developed product fulfils the requirements. It comprehends the static testing technique that involves the following activities:

- review; find and eliminate errors or ambiguities in documents such as requirements, design, test cases, etc.
- analysis; the code written by developers is analysed (mostly through the use of tools) for structural defects that may lead to errors.

Validation involves checking the software input values and analysing the output values. These processes comprise dynamic testing techniques that can include multiple levels:

- unit-testing: isolate each unit of the software to identify, analyze and fix the defects related to the developed code
- integration testing: verify the functional, performance, and reliability between the modules that are integrated
- system testing: black-box testing technique performed to evaluate the system's compliance against specified requirements in an end-to-end perspective
- acceptance testing: evaluate the system's compliance with the business requirements and verify if it has met the required criteria for delivery to end-users

As shown in Figure 2, verification and validation complement each other and the testing workflow starts from verification followed by the validation.

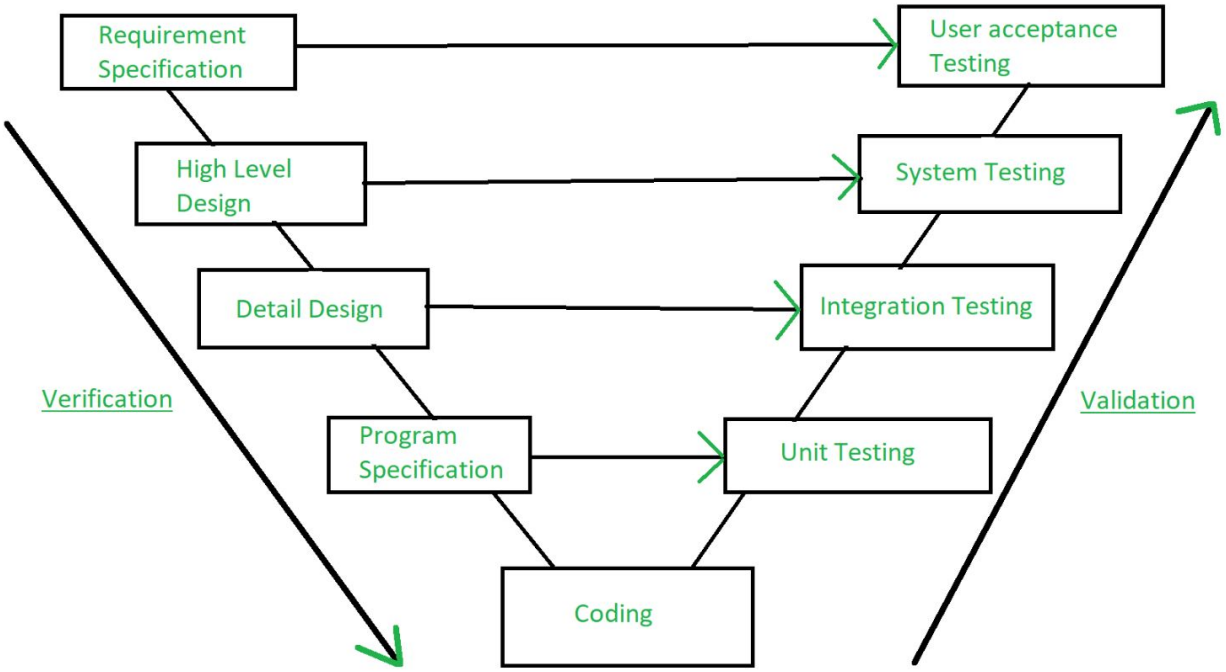


Figure 2: Verification is followed by Validation.

The CI/CD software engineering approach allows translating previous verification and validation workflow into a pipeline as code. The required environment is composed of a git repository manager that offers version control and source code management features, issue tracking, docker image registry and continuous integration and deployment pipeline service. This will allow to automate the verification and validation procedures, from the static tests, over the dynamic tests and ending with the release creation after the automated acceptance tests (Figure 3).

The pipeline as code describes a set of features that allows the pipeline service users to define pipelined job processes with code, stored and versioned in a source repository. These features allow users to discover, manage, and run jobs for multiple source repositories and branches, eliminating the need for manual job creation and management. With this pattern it is possible to deliver the resulting configuration to any repository where the workflow model should be applied.

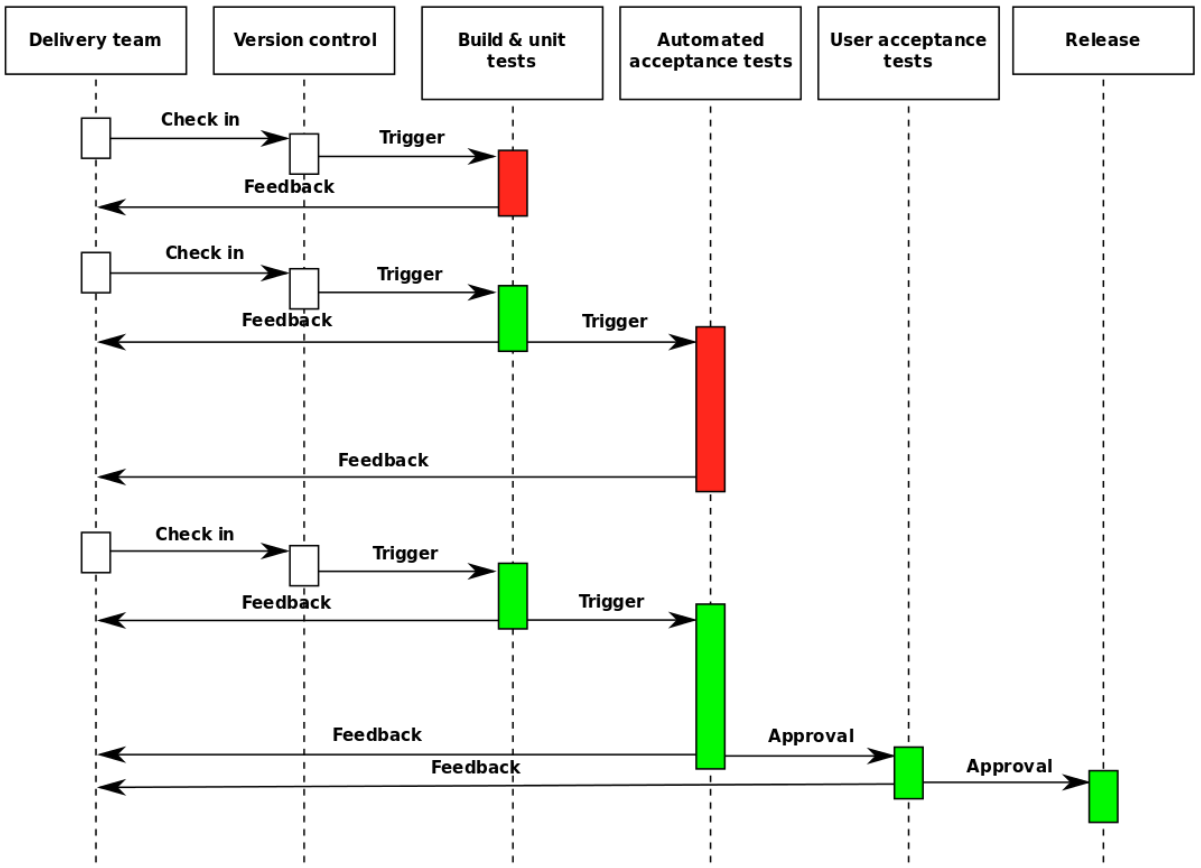


Figure 3: Sequence diagram describing the continuous delivery process.

3. Quality Assurance and Component Integration

Service development, provisioning, operation and integration into large scale production infrastructures often suffers from a lack of quality assurance realization. This may result from the fact that the different involved actors are either not aware of the benefits of applying quality practices, or not keen to adhere to them as they might increase the burden of service development, deployment and management. Thus, to understand the importance of these practices for software and services, this section introduces a *lightweight and practical approach* to educate and, ultimately, achieve a quality of those focused on stability or functional suitability.

A set of abstract Software Quality Assurance(SQA) criteria is detailed in [OLD+17]. It describes the quality conventions and best practices that apply to the development phase of a software component within the scientific communities. This set of criteria was created, detailed and applied in the software development process within several European projects. The criteria is abstract with respect to the services and tools that can be chosen to verify such criteria, while at the same time being pragmatic and focusing on automation of verification and following a DevOps approach.

The notational conventions in this document are to be interpreted as described in RFC 2119 [B+97]. Thus, the criteria described in this document have different levels of requirements.

Ideally for the BigHPC project, the development of each software component would follow the recommendations of this SQA baseline criteria, but having in mind that certain criteria may take a significant effort to implement, such as unit testing or functional testing. Taking into account that many components may be in an advanced state, it is perceived that at least the following Quality Criteria (QC) should be met:

- **Code Accessibility [QC.Acc]**: Following the open-source model, the source code being produced MUST be open and publicly available to promote the adoption and augment the visibility of the software developments.
- **Licensing [QC.Lic]**: As open-source software, source code MUST adhere to an open-source license [OSD] to be freely used, modified and distributed by others.
- **Code Workflow [QC.Wor]**: A change-based approach is accomplished with a branching model. Semantic Versioning specification [PW] is RECOMMENDED for tagging the production releases.

- **Code Management [QC.Man]**: Recommendation for the existence of an issue tracking system, to track down both new enhancements and defects (bugs or documentation typos). Pull or Merge requests provide a place for review and discussion of the changes proposed to be part of an existing version of the code.
- **Code Style [QC.Sty]**: Code style requirements pursue the correct maintenance of the source code by the common agreement of a series of style conventions. These vary based on the programming language being used. Each individual software product MUST comply with community-driven or de-facto code style standards for the programming languages being used.
- **Integration Testing [QC.Int]**: Integration testing refers to the evaluation of the interactions among coupled software components or parts of a system that cooperate to achieve a given functionality. Integration testing outcomes MUST guarantee the overall operation of the software component whenever new functionality is involved.
- **Documentation [QC.Doc]**: MUST exist, be publicly available, and specify how it should be managed and what types and formats should be used.
- **Security [QC.Sec]**: Secure coding practices SHALL be applied into all the stages of a software component development lifecycle, such as: compliance with Open Web Application Security Project secure coding guidelines [OWASP]; perform Static Application Security Testing [SAST]; or perform Dynamic Application Security Testing [DAST].
- **Code Review [QC.Rev]**: Code review MUST be done for all the developed code. Code review implies the informal, non-automated, peer, human-based revision of any change in the source code. It appears as the last step in the change management pipeline, once the candidate change has successfully passed over the required set of change-based tests.
- **Automated Deployment [QC.Aud]**: Production-ready code SHALL be deployed as a workable system with the minimal user or system administrator interaction leveraging Software Configuration Management (SCM) tools.

Besides the software, it is also important to take into account the services. With that goal in mind, the Common Service Quality Assurance Baseline Criteria [ODG+20], establishes the minimum viable set of quality requirements for an initial approach to Service Quality Assurance. The aim is to apply it in the integration process of the services existing under the BigHPC project. In order to discern among them, the RFC 2119 convention [B+97], is also used throughout the document, thus adding adequate information about the criticality of each requirement.

The “Common Service Quality Assurance Baseline Criteria” contextualizes what is a Service:

- **Web Service Definition** [WSD]:

    - 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 Definition** [WAD]:
    - A Web Application or "Web App" is a software program that is delivered over the Internet and is accessed through a web browser.
- **Platform or Service Composition** [SC]:
    - 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.

In a similar way as the SQA baseline, it is perceived that at least the following set of Service Quality Criteria [SvcQC] from [ODG+20], should be met:

- **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.
- **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.
- **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 [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). 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.

- **Policies [SvcQC.Pol]**: Policy documents describe what are the user's expected behaviour when using the Service, how they can access it and what they can expect regarding privacy of their data.
    - Acceptable Usage Policy (AUP)
    - Access Policy or Terms of Use
    - Privacy Policy
- **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 even documentation issues.
- **Automated Deployment [SvcQC.Aud]**: 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.
- **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.

BigHPC platform is a set of three main components (cf. deliverable 1.2):

- Monitoring
- Virtualization
- Storage

The Monitoring Component, besides providing the metrics that are consumed by the Virtualization and Storage Managers, it also supports the periodic testing of the services by

collecting a set of required metrics from the platform services. A status page provides an overview for the administrators and users about the APIs availability and manager components.

All components of the platform follow the proposed quality criteria baseline for both software and services. The implementation procedure of each component should be evaluated over the defined criteria. The expected result should be a list of criteria to be applied to each component.

Looking in more detail to each main component, and focusing on the most relevant criteria, the CI/CD pipeline would implement the required steps to provide the final SQA report for all services that comprise the platform (Monitoring, Virtualization and Storage); [SvcQC.Mon], [SvcQC.Aud], [SvcQC.Sec], [SvcQC.Int], [SvcQC.Pol], [SvcQC.Sup] and [SvcQC.Doc]. Regarding security [SvcQC.Sec], there is a significant number of criteria and it is perceived that it can take a significant effort to implement all tests on all services, as such the developers and the members of the WP5 will evaluate on a case by case basis at least a partial implementation of such tests.

The report provides an added value for any given release, while the pipeline provides an automated procedure to test the components and their integration. This provides an added trust when upgrading the platform as well as for the dissemination of the solution to other providers that plan to adopt the BigHPC platform.

The pipeline delivered to the providers should follow a Continuous Delivery (CD) strategy. This approach allows all types of changes; new features, configuration changes, bug and security fixes, to be deployed in the testbeds or production environment, in a safe, quick and sustainable way.

Regarding the software development of each component, it would be recommended the adoption of Continuous Integration (CI) practice to automate the integration of code changes for all project contributions. This is a DevOps best practice that allows the continuous contributions to be tested over a central repository, where developers code would be merged after validation. CI pipeline should focus on software baseline covering at least the following criteria: [QC.Sty], [QC.Int], [QC.Doc] and [QC.Sec]. It's a choice from the developers what other

criteria may be implemented in the pipelines of their components: [QC.Acc], [QC.Lic], [QC.Wor], [QC.Man] and [QC.Rev].

4. Pilot Testbed

The appropriate type of Pilot testbed infrastructure depends upon the two categories of components described in chapter 2: platform services and software components.

As such, the most appropriate infrastructure for the first category of components (i.e., Virtualization and Storage Manager, Monitoring services) are virtualized and highly customizable resources - Virtual Machines and/or containers including the management of such resources. The most appropriate type of resources for the second category of components, is a partition of an HPC cluster with the same environment as a production one - hardware, operating system, software, libraries and applications.

The pilot testbed serves three types of actors: Software developers, system administrators and end users or researchers.
In BigHPC, it is foreseen to have two pilot testbed infrastructures depicted in Figure 4:

1. Development testbed (light yellow box on the left): is more volatile or unstable, used by the software developers to test and integrate components or services at a fast pace.
2. Preview testbed (light green box on the right): to be used for software and service quality assessment and validation, for testing and preview by the platform system administrators and by end users to execute benchmarks and applications.

The pilot testbed type 1 platform services (c.f. section 2. Platform Integration), developed by BigHPC, can be deployed independently in the two testbeds, i.e. isolated from each other; these are the two boxes (in green and yellow) depicted in each testbed in Figure 4.
External services and components, not developed by BigHPC, but that are needed as integral part of the platform, may be common to both testbeds if that does not hinder the purpose of each testbed (blue box common to both testbeds in Figure 4). The Repository service is one such case, it's part of the Virtualization Manager and hosts the container images used to be executed in the HPC cluster.
Last, but not least, the low level software components that are deployed in the HPC cluster nodes are represented by the light red box on the lower part of Figure 4, and is common to both testbeds. An alternative may be considered if these components also need a more stable environment for the end users and system administrators, and another more fast paced

environment for the software developers, thus two partitions of the HPC cluster may be considered.

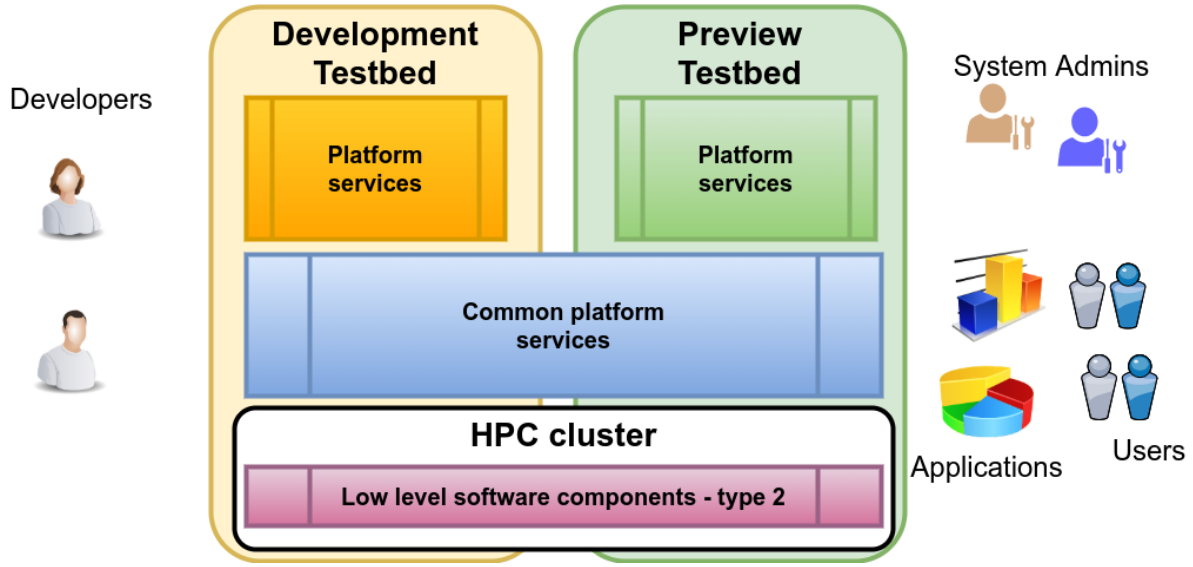


Figure 4: Pilot testbeds depicting the two foreseen infrastructures as well as how the components and services will be deployed.

Current components of BigHPC use both testbeds for the software development and service integration in the final platform. These are the references for testbed development in the first phase of the project.

On the side of the users application testing, the applications are packaged inside containers, and prepared to be run on the HPC infrastructures of TACC and MACC, examples of such applications are Gromacs, NAMD, Ambertools, Tensorflow or SPECFEM3D. Those application containers will be used in the Preview Testbed to test new versions of the BigHPC platform albeit with smaller inputs, or smaller problems, adequate for smaller cluster partitions available in the testbeds.

Furthermore, a set of benchmarks decided by the users or using as reference known HPC benchmark suites such as the PRACE Benchmark Suites [PBS], will be packaged into containers and executed in the testbed to test the low level software components alongside with the other applications.

The testing with real applications and benchmarks will allow further validation and gathering of feedback by the users, providing an environment to demonstrate the platform capabilities, both for users and for system administrators.

5. Conclusion

This deliverable describes the Pilot Testbeds of the BigHPC project. In particular, it proposes the architecture of the Testbeds and the number of infrastructures, their purpose and usage. The TACC and MACC HPC centers are the resource providers for the Testbed infrastructures.

The architecture of the Testbeds is aligned with the BigHPC platform architecture described in Deliverable 1.2. This document describes in a more abstract way the type of software components and services that are developed by the BigHPC project, as well as take into account external components and services that are needed to fully achieve the platform architecture.

This document also described who are the actors involved with the Testbeds as well as their roles; either as software developers or as system administrators and users (researchers).

The validation of the BigHPC platform is performed through a Software and Service Quality Assurance baseline criteria, aiming for higher quality of the developed Software and services as well as higher stability and sustainability.

It is deemed very important the Preview of the platform by both system administrators and by users, allowing them to execute both benchmarks and applications in a close to production infrastructure that is the Preview Testbed.

References

[FQJ+16] Fox, G., Qiu, J., Jha, S., Ekanayake, S., Kamburugamuve, S., 2016. Big Data, Simulations and HPC Convergence. Workshop on Big Data Benchmarks.

[MR+19] Milewicz, R., Rodeghero, P., 2019. Position Paper: Towards Usability as a First-Class Quality of HPC Scientific Software. IEEE/ACM International Workshop on Software Engineering for Science (SE4Science).

[HDM+15] Howison, J., Deelman, E., McLennan, M., Ferreira da Silva, R., Herbsleb, J., 2015. Understanding the scientific software ecosystem and its impact: Current and future measures. Research Evaluation, Volume 24, Issue 4, October 2015, Pages 454–470.

[OLD+17] Orviz, Pablo; López García, Álvaro; Duma, Doina Cristina; Donvito, Giacinto; David, Mario; Gomes, Jorge, 2017. A set of common software quality assurance baseline criteria for research projects. CSIC-UC - Instituto de Física de Cantabria (IFCA). http://hdl.handle.net/10261/160086.

[B+97] S. Bradner, Key words for use in RFCs to Indicate Requirement Levels, https://www.rfc-editor.org/info/rfc2119 .

[OSD] The Open Source Definition, URL: https://opensource.org/osd.

[PW] Tom Preston-Werner, Semantic Versioning 2.0.0, URL: https://semver.org.

[ODG+20] Orviz, Pablo; David, Mario; Gomes, Jorge; Pina, Joao; Bernardo, Samuel; Campos, Isabel; Moltó, Germán; Caballer, Miguel, 2020, A Set of Common Service Quality Assurance Baseline Criteria for Research Projects. CSIC-UC - Instituto de Física de Cantabria (IFCA). http://dx.doi.org/10.20350/digitalCSIC/12533.

[WSD] Web Service Definition https://techterms.com/definition/web_service

[WAD] Web Application Definition https://techterms.com/definition/web_application

[SC] Service Composition - Glossary CSRC
https://csrc.nist.gov/glossary/term/Service_Composition

[OWASP] The OWASP Foundation, OWASP Secure Coding Practices-Quick Reference Guide
https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/migrated_content.html
[SAST] The OWASP Foundation, Source Code Analysis Tools
https://owasp.org/www-community/Source_Code_Analysis_Tools
[DAST] The OWASP Foundation, Dynamic Application Security Testing, URL:
https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools
[PBS] PRACE Benchmark Suites, URL:
https://prace-ri.eu/training-support/technical-documentation/benchmark-suites/