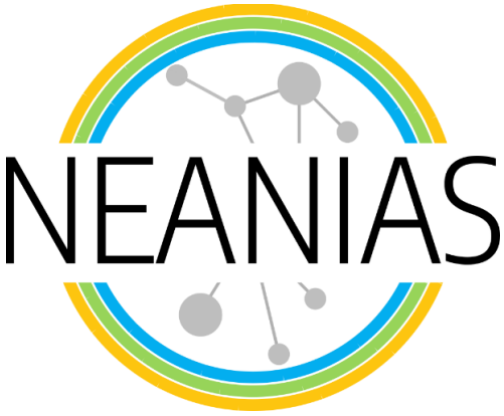

NEANIAS
Novel EOSC services for Emerging Atmosphere, Underwater and Space Challenges

Deliverable

Deliverable: D7.1 Delivery Activities Methodology and Plan

31/03/2020

NEANIAS is funded by European Union under Horizon 2020 research and innovation programme via grant agreement No. 863448

Document Info

Project Information			
Acronym	NEANIAS		
Name	Novel EOSC Services for Emerging Atmosphere, Underwater & Space Challenges		
Start Date	1 Nov 2019	End Date	31 Oct 2022
Program	H2020-EU.1.4.1.3. - Development, deployment and operation of ICT-based e-infrastructures		
Call ID	H2020-INFRAEOSC-2018-2020	Topic	H2020-INFRAEOSC-2019-1
Grant No	863448	Instrument	RIA

Document Information			
Deliverable No	D7.1		
Deliverable Title	Delivery Activities Methodology and Plan		
Due Date	31-Mar-2020	Delivery Date	10-Apr-2020
Lead Beneficiary	GARR (18)		
Beneficiaries (part.)	GARR (18), NKUA (1), INAF (3), CITE (7), UOP (8), UBREMEN (10), ALTEC (19)		
Editor(s)	Claudio Pisa (GARR)		
Author(s)	Claudio Pisa (GARR), Georgios Papanikos (CITE), Konstantinos Kakaletris (CITE), Eva Sciacca (INAF), József Kovács (SZTAKI), Attila Farkas (SZTAKI), Nikos Chondros (NKUA), Simone Mantovani (MEEO)		
Contributor(s)	Cristobal Bordiu (INAF), Carmelo Manetta (ALTEC), Mel Krokos (UOP), Marco Lorini (GARR)		
Reviewer(s)	George Papastefanatos (ATHENA), Christina Peraki (ATHENA)		
Workpackage No	WP7-Delivery		
Version	V1.0	Stage	Final
Version details	Revision: *166* . Last save: *2020-04-10 , 08:18* Pages: *53* . Characters: *81.357*		
Distribution	Public	Type	Report
Keywords	Delivery, EOSC, Development, Service Operation, Quality metrics, Service Management, Software Development		

Change Record

Version	Date	Change Description	Editor	Change Location (page/section)
1.0	10/04/2020	Document version submitted to EC	Claudio Pisa	

Disclaimer

NEANIAS is a Research and Innovation Action funded by European Union under Horizon 2020 research and innovation programme, via grant agreement No. 863448.

NEANIAS is project that comprehensively addresses the 'Prototyping New Innovative Services' challenge set out in the 'Roadmap for EOSC' foreseen actions. It drives the co-design, delivery, and integration into EOSC of innovative thematic services, derived from state-of-the-art research assets and practices in three major sectors: underwater research, atmospheric research and space research. In each sector it engages a diverse set of research and business groups, practices, and technologies and will not only address its community-specific needs but will also enable the transition of the respective community to the EOSC concept and Open Science principles. NEANIAS provides its communities with plentiful resource access, collaboration instruments, and interdisciplinary research mechanisms, which will amplify and broaden each community's research and knowledge generation activities. NEANIAS delivers a rich set of services, designed to be flexible and extensible, able to accommodate the needs of communities beyond their original definition and to adapt to neighboring cases, fostering reproducibility and re-usability. NEANIAS identifies promising, cutting-edge business cases across several user communities and lays out several concrete exploitation opportunities.

This document has been produced receiving funding from the European Commission. The content of this document is a product of the NEANIAS project Consortium and it does not necessarily reflect the opinion of the European Commission. The editor, author, contributors and reviewers of this document have taken any available measure in order for its content to be accurate and lawful. However, neither the project consortium as a whole nor the individual partners that implicitly or explicitly participated in the creation and publication of this document may be held responsible for any damage, financial or other loss or any other issue that may arise as a result of using the content of this document or any of the project outputs that this document may refer to.

The European Union (EU) was established in accordance with the Treaty on the European Union (Maastricht). There are currently 28 member states of the European Union. It is based on the European Communities and the member states' cooperation in the fields of Common Foreign and Security Policy and Justice and Home Affairs. The five main institutions of the European Union are the European Parliament, the Council of Ministers, the European Commission, the Court of Justice, and the Court of Auditors (http://europa.eu.int/).

Table of Contents

Tables of Figures & Tables

Document Figures

Document Tables

Abstract

NEANIAS aims at contributing to the materialization of the European Open Science Cloud (EOSC) by delivering innovative thematic services in the Underwater, Atmospheric and Space research sectors.

This document focuses on recommendations and procedures to effectively deliver, operate and monitor NEANIAS services, to integrate them with the EOSC ecosystem and to access the related facilities and infrastructures.

The service management processes that are here proposed are based on the FitSM standard, and address the management of the NEANIAS service portfolio, related agreed service levels, software release processes, as well as incidents and customer relationship.

Tools and guidelines are then provided to support software development, testing, release, deployment and documentation activities. To the same aim, software licensing issues are also covered.

To support NEANIAS services during their operation, processes and tools covering feedback and issue tracking are described, along with monitoring, alerting and quality metrics gathering facilities and guidelines.

Finally, a work plan regarding the delivery of the NEANIAS services in the following months is reported.

1. Introduction

The NEANIAS project aims at delivering TRL8 (i.e. system complete and qualified) innovative services in the Underwater, Atmospheric and Space research sectors to contribute to the European Open Science Cloud (EOSC).

To effectively deliver, operate and monitor such services, NEANIAS relies on facilities provided by the NEANIAS consortium or by single NEANIAS partners, accessed and utilised through procedures and guidelines described in this document. The requirements for these facilities are based on well-established practices and adopting mature tools, with directions dictated by both the adopted Service Management schemes reported below and an internal survey within NEANIAS Service Providers (Appendix I). Moreover, this document defines the methodologies deployed for managing the software codebase lifecycle (i.e. development, building, testing, licensing, releasing and monitoring) along with proposed quality metrics concerning the operation of the NEANIAS services.

In Section 2 a description of the FitSM [1] service management processes, which we are employing in NEANIAS, is reported. Then software implementation facilities, along with procedures, methodologies and guidelines, used to implement the FitSM RDM [3] process, are described in Section 3. Section 4 reports on procedures and guidelines concerning software documentation, while Section 5 provides source code licensing recommendations. Section 6 reports on facilities to supply effective feedback and resolve issues concerning NEANIAS services source code and operations, to implement the Customer Relationship Management (CRM) and Incident and Service Request management (ISRM) FitSM processes. Section 7 addresses monitoring and alerting facilities, along with a description of the recommended quality metrics to be collected. Section 8 describes the infrastructures employed to enable the NEANIAS services provisioning as well as infrastructure specific access and usage procedures. Section 9 provides general recommendations and remarks concerning service implementation and operation. Finally, a work plan of the NEANIAS delivery activities is proposed in Section 10.

2. Service Management Processes

The consortium is going to provide eInfrastructure services for the NEANIAS service portfolio during the project. We will define a plan on how to support EOSC processes with our eInfrastructure resources and in strong cooperation with WP8, we will implement the EOSC integration concerning the FitSM service management processes.

Several standards and frameworks for service management exist (most notably, ITIL [26] and ISO/IEC 20000); NEANIAS will follow the standard FitSM, which is largely adopted in EU research projects. FitSM is a free and lightweight [27] standards family for effective IT service management in a wide range of organisations. FitSM provides an achievable standard for effective IT service management (ITSM) via manageable documentation and sustainable principles for the service management processes. These processes do not generate unreasonable overhead for service management. FitSM also provides solutions which can be efficiently implemented in organisations ranging from SMEs and start-ups to large enterprises. The following processes, related to the FitSM-1 [3], provide a set of requirements for IT Management, and they are especially tuned to the needs of federated infrastructures or groups new to ITSM. These processes will cover the portfolio management and agreed service levels of the provided services. Also provides guidelines for release management, unplanned incident management and good customer relationship management as well.

2.1. Service Portfolio Management (SPM)

The main goal of SPM is to define and maintain a service portfolio. SPM is an internal list that details all the services offered by a service provider, including those in preparation, live and discontinued. This service portfolio will be the basis of the service catalogue which will be shared with the customer. The service portfolio shall be maintained during the whole project and all services shall be specified as a part of the service portfolio. The design and the release process of a new or changed service should be planned and these plans shall consider the timescales, responsibilities, new or changed technology, communication and service acceptance criteria. Finally, the organizational structure which will support the delivery methods should be identified as well. The defined service portfolio should be maintained during the long-term sustainability period after end of the project.

2.2. Service Level Management (SLM)

The main purpose of SLM is to maintain a service catalogue, and to define, agree and monitor service levels with customers by establishing meaningful and sustainable service level agreements (SLAs) and operational level agreements (OLAs) and underpinning agreements (UAs) with suppliers.

Definitions from FitSM-0 [2]:

- Service level agreement (SLA): Documented agreement between a customer and service provider that specifies the service to be provided and the service targets that define how it will be provided. [29][30]
- Operational level agreement (OLA) Agreement between a service provider or federation member and another part of the service provider's organisation or the

federation to provide a service component or subsidiary service needed to allow provision of services to customers.

- Underpinning agreement (UA) Documented agreement between a service provider and an external supplier that specifies the underpinning service(s) or service component(s) to be provided by the supplier, and the service targets that define how it will be provided.

The main part of the SLM is to produce a service catalogue for customers and agree on the SLAs with customers as well. SLM also includes the agreement of the OLAs and UAs with the supporting parties and suppliers to ensure that the service target SLAs can be met. Finally, the evaluations of the current service performances like response time is necessary to meet the defined SLAs.

2.3. Customer Relationship Management (CRM)

The main objective of CRM is to establish and maintain a good relationship with customers receiving services. Under the CRM processes, it is defined that there shall be a designated contact responsible for each customer to manage the customer relationship and satisfaction. Service reviews with the customers shall be conducted at planned and regular intervals. Customer complaints shall be managed on per service and customer basis. Finally, customer satisfaction shall be managed via different approaches like responding to customer requests and resolving any customer related issues within limited timeframes.

2.4. Release & Deployment Management (RDM)

The main purpose of RDM is to bundle changes of one or more configuration items to releases so that these changes can be tested and deployed to the live environment together. During this process, the whole release procedure has to be defined for each service. The deployment of new or changed services and service components to the live environment shall be planned with all relevant parties including affected customers as well. The releases shall be built and tested in a test environment to resolve any issues before deployment into the production environment. Finally, the deployment preparation shall consider steps to be taken in case of unsuccessful deployment to reduce the impact on services and customers. The following process, which is presented in Figure 1 - RDM workflow should be planned during the RDM process. The whole RDM workflow starts with some published change in the current source code via an incoming pull request, for example. This request will be registered and classified in the version control system. After the classification, the developers will check and hopefully approve these changes and merge them into the main source code. After the merging, the Release & Deployment processes will be executed as presented in Figure 1 - RDM workflow [3]. During these processes, the modified source code will be built and tested automatically and if every test is passed, then deployed into the production system during the next release period. After this whole process, the newly released version will be revised to provide feedback for the original pull request.

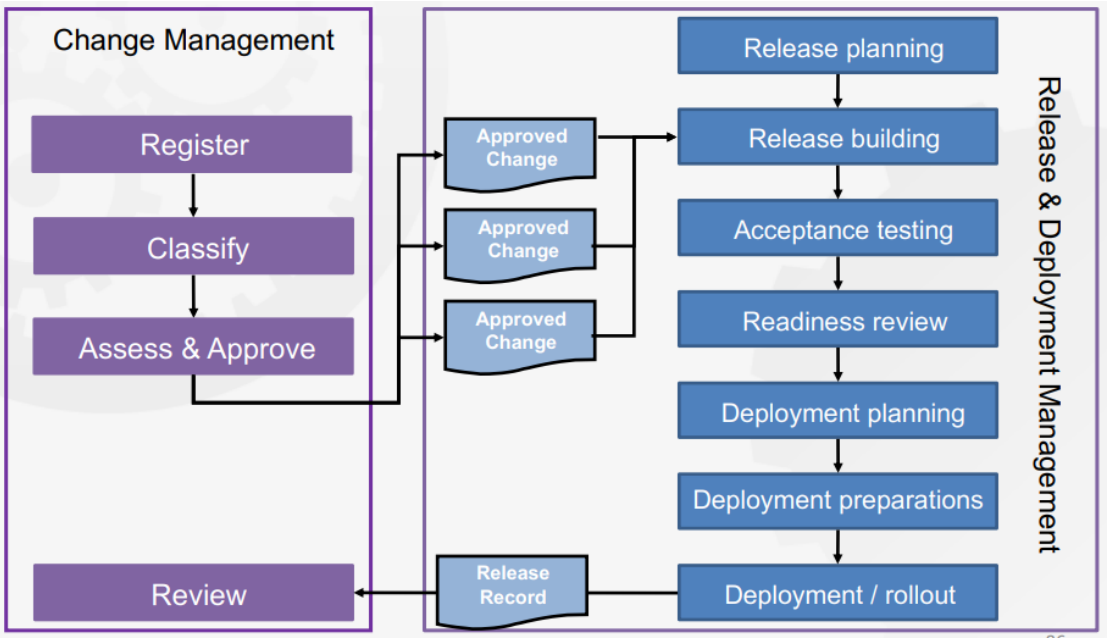

Figure 1 - RDM workflow [3]

2.5. Service Availability and Continuity Management (SACM)

The main goal of SACM is to ensure sufficient service availability to meet agreed requirements and adequate service continuity in case of exceptional situations like unplanned hardware or service failure. The most important output of this process is a service availability and continuity plan for every defined service in the service portfolio. During this process, the requirements have to identify for service availability and continuity based on the agreed SLAs. This process has to produce a plan to reduce the probability and impact of identified availability and continuity risks. Finally, the main procedure to support these plans is to monitor the service availability during the whole service lifecycle.

2.6. IT Security Management (ISM)

The main purpose of the ISM is to manage information security effectively through all activities performed to deliver and manage services so that the confidentiality, integrity and accessibility (CIA) of relevant assets are preserved. The necessary security level for these services will be observed and specified per application contexts not generally for every application. The elements of the CIA model shall be managed during this process. The threats and risks associated with the stored information will also need to be identified. The most important output of this process is the information security policies and information security risk assessments.

2.7. Incident and Service Request Management (ISRM)

The main objective of ISRM is to restore normal or agreed service operation within the agreed timeframe after the occurrence of an incident and to respond to user service requests. During this process, every incident and service request shall be registered, classified and prioritised

in a consistent manner. The defined SLA shall be the baseline for the prioritization of incidents and service requests. Every involved person in the incident management process shall have access to relevant information including known errors, workarounds, configuration and release information. Within this process, major incidents and consistent approaches to manage them will be identified. Finally, the users shall be kept informed of the progress of incidents and service requests they have reported.

3. Software Implementation

In this section, we introduce some of the key aspects of the Release and Deployment Process as described by the selected ITSM.

3.1. Software Implementation Support System

A Gitlab deployment is available for the members of the NEANIAS consortium at the URL https://gitlab.neanias.eu. Gitlab [5] is an open source web-based platform which allows software developers to manage git source code repositories providing them with issue tracking, wiki and CI/CD pipeline facilities. Gitlab can also support the publication of artifacts through package and docker registries. This allows for an integrated environment for source code management, automated building, automated testing and publishing.

As described in the following sections, the Gitlab platform will not be the only tool to be employed in NEANIAS: indeed, several services to be deployed during the project are already successfully using their own software implementation support systems. Also, intellectual property issues may arise (e.g. service U2 “Seafloor Mosaicking from Optical Data” is using a non-FOSS license). Thus, to foster sustainability, hosting services’ source code on the Gitlab platform is only a recommendation, which applies especially to new services to be developed during the NEANIAS project timeframe.

3.2. Repository Structure

In this section, a branching strategy is recommended mainly for software coding in newly built repositories, however it can be applied to already existing repositories as well.

The Git version control system [22] allows for the creation of lightweight development branches, which may cause branch proliferation in code repositories, which in turn can lead to errors and render code development activities difficult to follow. Adopting instead a coherent branching and tagging strategy may improve code quality and increase code production speed.

A branching strategy is also important to provide support for multiple, parallel developments and to separate stable code from in-progress or broken code. When multiple developers are working on the same code, a branching strategy can help in organizing the progress of the different developments. The outcome of the developments goes into different branches, which later can be tested separately and combined freely by the repository owner (e.g. which feature to be released). The importance of the branching strategy is increasing with the size of the repository.

The software repository must contain a **master** branch which contains all the code changes (commits) during the lifetime of the software. This branch contains commits which result in a stable and healthy version. The master branch usually contains the releases of the software. A testing strategy must be designed in order to make sure only error-free code is introduced in the master branch.

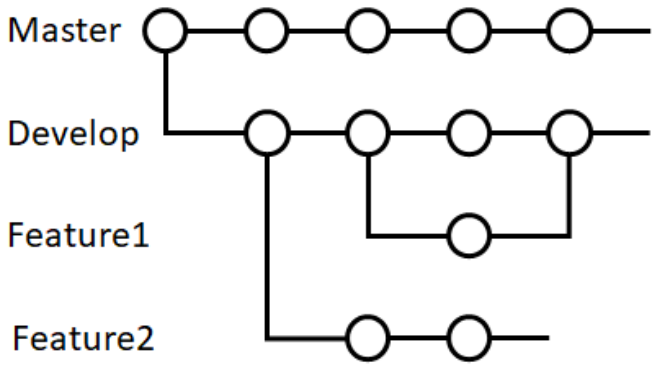

Figure 2 - Source Control Branching Strategies

Tagging may be applied in order to mark a version. Tagging is supported by gitlab. Releases may have a distinguished tagging procedure on the master branch. Tagging may be applied at any time and at any point in the branches, however in most cases, release procedures require it.

The application repository should have a long-living branch called **develop** that runs alongside the master branch. All work is committed first to the develop branch. This is a safe place to introduce new code that requires further testing. The software code in the develop branch must be tested intensively before merging back to the master. Merging is usually scheduled at the end of release procedures.

Whenever a new feature is to be developed, a new **feature** branch should be created. All the changes related to the feature must be committed here. When the feature development is finished, it is merged back to the develop branch. Several feature branches may exist in parallel. At this point, code review may be applied, the owner of the repository may decide on it.

As for the repository structure, the recommendation, which applies to software projects which start during the timeframe of the NEANIAS project, is to provide in the root of the git repository the following items:

- a README file (in plain text, markdown or restructured text format) containing a description of the repository contents and a pointer to the software documentation;
- a LICENSE file specifying the software license and terms of use (or pointing to this information);
- a MONITORING file describing the provided monitoring endpoints (or pointing to the documentation);
- a src/ directory, containing the actual source code and tests.

To foster sustainability, software which is pre-existing to NEANIAS and which is already using its own development practices and conventions should ignore the guidelines described above in this section.

3.3. Software Building, Deploying and Testing

3.3.1. Deploy & Upgrade methodology

The deployment of NEANIAS services is envisaged to be based on automatic mechanisms and to follow the Infrastructure as Code (IaC) paradigm (§ Section 9.2), to foster service reproducibility at different times and on different infrastructures. To this aim, the use of text-based automation tools, as the ones in the list reported below, is highly encouraged. This is extended also to other tools which are not included below, but still allow automation and are text based. Please note that the same tools can be used for service upgrade, although this may require a different path in the installation process.

Ansible [8] allows developers to create textual descriptions of complex deployments. These, which are called *playbooks* in the Ansible terminology, can be executed automatically against a set of target machines to configure them. Ansible playbooks can be effectively versioned through git and are idempotent, i.e. subsequent executions of the same playbook leave the target machine(s) in the same desired final state.

Kubernetes [9] is a platform for container orchestration which automates the deployment, scaling and upgrade of applications, while providing abstractions for underlying resources. It is based on textual descriptions, which can be versioned through git, which define the system desired final state and which are presented to the Kubernetes API. Then the Kubernetes control plane components continuously compare the state of the system with the desired state and perform the required operations.

Docker Compose [10] allows developers to define applications by connecting multiple containers. It is based on simple textual definitions and can be effectively used to build, deploy and upgrade services.

Occopus [11] is a cloud orchestrator tool to deploy complex infrastructures in clouds. It is a framework that provides automatic features for configuring and orchestrating distributed applications (so called virtual infrastructures) on single or multi cloud systems. For applications requiring multiple components to be deployed and orchestrated on multiple virtual machines, Occopus is a good solution. The tool is being developed by a consortium member, so feature requests can be handled in a very efficient way.

3.3.2. NEANIAS testing and building infrastructure

Gitlab has built-in support for Continuous Integration / Continuous Delivery (CI/CD) workflows [6]. This provides automated testing and building facilities for the NEANIAS services, in order to detect errors starting at the software development stages. Thus, the use of automated testing and building facilities is highly recommended for NEANIAS service development.

Using the Continuous Integration (CI) technique, each time a set of changes is pushed to a repository, the code is built and tested automatically. This allows to detect issues early on and avoid programming on top of buggy code.

With Continuous Delivery (CD) the artifacts are also automatically prepared to be deployed, although the actual deployment step is still triggered manually. Continuous Deployment automates also this last step.

Build scripts, unit and integration tests, as well as deployment scripts can be defined in Gitlab through a set of YAML definitions and files. The scripts are grouped into jobs, which can compose pipelines. These are executed by Gitlab runners on the specified conditions and events. For technical details we refer to the Gitlab documentation at [7].

3.4. Software Releases

This section presents the different approaches proposed to the NEANIAS software developers to be deployed with software releases of the NEANIAS services including code versioning and release through Docker and Singularity containers as well as Virtual Machine images.

3.4.1. Code versioning

Service software releases are proposed to be managed by the GitLab Tags feature. Git has the ability to tag specific points in a repository's history as being important. Typically, those are used to mark release points (v1.0, v2.0 and so on). Two types of tags are supported: lightweight and annotated. A lightweight tag is very much like a branch that does not change — it is just a pointer to a specific commit. Annotated tags, are stored as full objects and are checksummed, they contain the tagger name, email, date, a tagging message and can be signed and verified with GNU Privacy Guard (GPG). Major versions should report detailed release notes summarizing the new features and any other main important changes. It is generally recommended to create annotated tags to get all this information; but for temporary tags, lightweight tags are preferable.

3.4.2. Docker Containers

With the Docker Container Registry integrated into GitLab, every project can have its own space to store its Docker images. GitLab offers a simple Container Registry management panel. The panel view shows all Docker images in a project and can easily allow users to manage them. While it is possible to build and push images from the user's local machine, the true power of the Container Registry comes when it is combined with GitLab CI/CD. It is possible to create workflows and automate processes that involve testing, building, and eventually deploying projects from the Docker image created.

3.4.3. Singularity Containers

Singularity [31] is a container platform aimed at the scientific community, providing reproducible containerization for scientific computing and HPC applications.

GitLab makes it easy to build Singularity containers directly from GitLab repositories and so the intended use case is to use GitLab for version control and building, and then pull the artifacts using the Singularity Registry client.

The GitLab repository will need a file called `.gitlab-ci.yml` (that defines the jobs to run) to build the Singularity container. The most important components needed to pull the containers are:

- collection url is the GitLab repository name.
- job_name provides the different sections that correspond to jobs. This is for example the one used for building

```
build:
```

```
script:
   - /bin/bash .gitlabci/build.sh Singularity
```

- job_id: is a number that is associated with the particular job.
- tag: The tag for the container in Singularity Hub, is represented in the recipe name in the repository. A recipe called “Singularity” corresponds to the tag “latest,” while a file in the format “Singularity.thetag.simg” corresponds to tag “thetag.”

The Singularity Registry GitLab client provides the following commands:

- pull: [remote->local] pull layers from Docker Hub to build a Singularity image, and save in storage.
- search: [remote] search your personal Dropbox for a container

Note that there is not a push command, this is because of GitLab’s built in continuous integration (CI) which is directly built from a repository.

3.4.4. Virtual Machine images

Apart from Docker and Singularity containers, NEANIAS services may be delivered using Virtual Machines (VM). VM configuration files (e.g. Vagrant files) can be stored in GitLab while Images of VMs will be then released to Open Stack Glance service [25]. The Glance service allows users to upload and discover data assets that are meant to be used with other services including VM images and metadata definitions. Glance image services include discovering, registering, and retrieving virtual machine (VM) images. It has a RESTful API[1] that allows querying of VM image metadata as well as retrieval of the actual image. Glance hosts also a metadefs catalog that provides a way to programmatically determine various metadata key names and valid values that can be applied to OpenStack resources.

3.4.5. Packages

As some of the artifacts generated by the Release and Deployment process may have the form of software packages, the recommended way to store them and make them available for deployment is to leverage the OpenStack object store service (Swift). This service exposes a RESTful API [24] to upload, retrieve and manage files and directories abstracted as objects and object containers. Additionally, package managers at component level (such as Maven or npm) will be strongly encouraged.

3.4.6. Software Release Management

The activities of bundling all needed components, service, configuration and respective dependency definitions comes together in the process of packaging and defining a consistent, named software release. The elements that contribute to the delivery of one or more services (Configuration Items) [2] are bundled into releases so that any changes can be tested and deployed to a live environment along with their dependencies.

This process of defining a release is created to be standardized and repeatable in all its phases, including the planning, scheduling, bundling, testing, documenting and deploying aspects. Based on the included changes the release is marked as major, minor or an emergency release.

[1] https://docs.openstack.org/api-ref/image/

The rules by which releases are scheduled are governed by release policies, including the acceptance criteria by which a component or a specific change is made part of the release.

The continuous iteration over which the Release and Deployment process [4] is executed, defines at a minimum the following steps:

- Release Planning
- Build the release
- Run all testing scenarios on the new release
- Ensure all needed documentation and training material is available
- Prepare target environment for the deployment
- Alert and deploy
- Monitor and review release impact and success

4. Software Documentation

The NEANIAS services aim to be used by a wide range of users under the NEANIAS framework as well as the EOSC ecosystem. To facilitate the usage, full comprehensive and well-maintained documentation that covers multiple aspects of the service is fundamental.

4.1. Single Entry Point

To facilitate ease of discovery and consistent approach in the provided documentation, a single entry point for all documentation will be provided. NEANIAS will create a root level project at the popular and widely used *Read the Docs* online document sharing environment (https://readthedocs.org/).

The online documentation made available will be under the readthedocs.org domain but there is also the option to define a custom domain under neanias.eu, e.g. docs.neanias.eu, so that users reading the documentation easily identify the scope of documentation.

4.2. Documentation Source

It is not necessary that the documentation of all services share a single source where they are maintained and updated. In contrast, to allow easier management, versioning policies and possible access restriction policies, it is suggested that each service defines a different documentation source to maintain and manage its documentation. Each of these repositories will be bound as sub-projects under the single NEANIAS root documentation project.

The repositories where the documentation source is maintained can be any of the platform's supported source control systems. Within the NEANIAS centrally offered source control management system, dedicated repositories will be available for each service, where the respective service provider will be able to document their services. Additional documentation may exist in other platforms and external repositories. Still some information and at least linking content will be available under the NEANIAS root structure and further information links can be provided to the externally hosted content.

4.3. Content and Structure

For each service, depending on the scope, target audience and expected usage, different content may be desired by its users, required by system administrators or expected by system integrators and developers. No single guideline is imposed on service providers with respect to documentation they make available. Still, for each service, the following sections are expected to be covered to the respectively appropriate degree:

- NEANIAS ecosystem fit – A general description of the service and how it fits into the NEANIAS ecosystem of services
- User Manual – Information for the end user on how to use the service. Assuming a web interface for the functionality of the service is available, screenshots and respective narrative is encouraged
- Developer Documentation – All needed information with respect to exposed models, functionality and extension points

- API docs – If a public API is available for the service as integration points with external services, the description of the API and the flow of common operations. The API documentation can be made available using the OpenAPI specification [28], in which case, if publicly accessible, links to it can be included

Given the scope and target audience of this documentation, Administration, Configuration and Deployment documentation will not be hosted under the same repository and will not be exposed through the same entry point. Still, it is expected that each service will document all needed information on these aspects of the Service Lifecycle and will make it available as needed.

4.4. Versioning

As NEANIAS services evolve, so will the available documentation. It is therefore required that it will be easily possible for a user to browse through different versions of the documentation and all service releases can be easily linked to the respective documentation version.

Semantic Versioning [20] will be used for the documentation and it is proposed that tags on the documentation source repository mark the respective version of the documentation.

4.5. Automations

To streamline documentation publishing and ensure that the most up-to-date content is always available, the available automations can be employed. Webhooks will be defined at the central NEANIAS documentation repository and subsidiary service projects. A consistent approach with respect to version tagging and branch usage will be followed across service documents to facilitate the automation.

4.6. Internationalization

The target audience of NEANIAS Services is widespread, at minimum, across Europe. Still, as a baseline for all available documentation, the English language is set. Any additional localizations are encouraged but not expected.

4.7. Privacy

All documentation made available through the central NEANIAS root project will use a public privacy level. Private or confidential documentation, if any, will not be supported or hosted.

4.8. Known Errors Database (KEDB)

For each service, a list of shortcomings, defects or problems may be known either from the release time or reported through respective ticket incidents. Along with the proper documentation, a known error data base needs to be maintained with these issues, shortcomings, planned fixes and known workarounds. This will act as a central knowledge center for users and a reference point when NEANIAS service providers respond to support requests.

4.9. Frequently Asked Questions (FAQ)

Parallel to the Known Errors Database that primarily focuses on tracking known workarounds for common and repeatable problems until they are resolved, a Frequently Asked Questions

section can facilitate users in getting some early answers and guidelines on common questions they may have. These can cover individual service operation, means of access as well as overall topics. This section can be maintained as part of the overall NEANIAS documentation as well as provide further linking to individual service material or even external references (e.g. EOSC).

5. Software Access & Licensing

5.1. Project's Access

GitLab allows developers to set a project's visibility as public, internal, or private. Public projects can be cloned without any authentication. They will be listed in the public access directory (/public) for all users. Any logged in user will have Guest permissions on the repository. Internal projects can be cloned by any logged in user. They will also be listed in the public access directory (/public), but only for logged in users. Any logged in user will have Guest permissions on the repository. Private projects can only be cloned and viewed by project members (except for guests). They will appear in the public access directory (/public) for project members only.

5.2. Access to Code that can be Shared

Github [12] is a popular site for open source code publishing, which makes it easy for other developers to contribute. On Github, projects can gain visibility as well as contributions from the wider community.

On the other hand, Gitlab [5] provides software (namely Gitlab CE) to manage code repositories on premises, which allows for facilities for automatic testing and building (which are available on Github only with premium subscriptions), and which allows for the hosting of code which is subject to IP restrictions.

To get the advantages of both approaches, Gitlab provides a feature called "repository mirroring" [13]: Gitlab repositories can be configured to synchronize specific git branches to (public) GitHub repositories. This allows to keep the main development workflow inside the on-premises facilities while enjoying the advantages of being "showcased" on GitHub.

5.3. Recommended Licenses

To foster the transition to Open Science, the recommendation is to employ copyleft, Free and Open Source Licenses for the NEANIAS service software wherever applicable. A list is available at [32] and include, among the most popular, the GNU General Public License [33], the Apache License [34] and the MIT license [35]. Developers should be aware that some of these licenses require or recommend to include licensing information also inside the source code itself, following the license specific guidelines.

5.4. Access to non-FOSS Software

Closed source software employed in NEANIAS should still be referenced inside the NEANIAS Gitlab. To this aim, partners should create a new project, using the name of the closed source software component, filling in the project description and choosing to initialize the repository with a README. Then the README.md file should be modified to include a summary of the component purpose and functionalities, software licensing information and links to existing documentation or relevant websites.

Moreover, binary artifacts such as packages, container images or virtual machine images should be made available for deployment purposes, following the recommendations in Section 3.4.

5.5. Integration with Third Party Services

Third party services, such as EOSC core services and OpenStreetMap tile providers, are planned to be leveraged by NEANIAS services. In this case, partners which are integrating these services with NEANIAS services should check if there are any license compatibility issues[2] both at the code level and at the API level. Also, pricing issues should be considered where applicable.

[2] See e.g. https://en.wikipedia.org/wiki/License_compatibility

6. Supplying Feedback and Resolving Issues

This section describes how we plan to implement actions that are targeted by the CRM and ISRM FitSM processes.

6.1. Tooling and Scope

Within NEANIAS, the following three categories of issue and service request sources have been identified. For each one, the main differentiators include the target users of the issue tracking service and the scope of the incident and service request:

- Help Desk - which targets external users of the NEANIAS services
- Inter-Service - High level / project wise development / integration / coordination tasks
- Intra-Service – Within a service itself, lower level development activities may be organized in partner specific ticketing systems

6.1.1. Help Desk

To assist and support the end users of the NEANIAS offered Services, a Help Desk needs to be set up that will act as the entry point for service consumers to report issues, incidents and requests in a structured fashion. The Help Desk main purpose is to collect:

- Incident Reports - Unplanned disruption of operation in a service or service component, or degradation of service quality versus the expected or agreed service level or operational level according to service level agreements (SLAs), operational level agreements (OLAs) and underpinning agreements (UAs) [2]
- Service Requests - User request for information, advice, access to a service or a pre-approved change [2]

For this purpose, a new space within the NEANIAS ticketing system will be created to serve as the single point of reference for Help Desk related activities. It is planned that this will be available under: https://ticketing.neanias.eu/projects/neanias-helpdesk

In addition to its main goals, the Help Desk may be used to achieve an additional set of Service Management processes to facilitate exploitation and enhance the offered service value. One of them is to establish and maintain a good relationship with customers receiving services (Customer Relationship Management) [4]. Although the underlying tool that supports the operation of the Help Desk will not be used as a full-fledged Customer Relationship Management tool, it can assist the process. Through the Incident and Service Requests, it will:

- Assist in maintaining contact points with the service customers
- Manage customer complaints – Record, handle and close complaints, monitor the actions and follow-up implementation of a customer complaint, periodically review status
- Manage customer satisfaction – Plan and prepare surveys, record survey results, initiate follow-up actions in response

To handle cases of escalating a ticket originally reported in the HelpDesk to some other Intra/Inter-Service issue tracking system, each ticket will be, at a minimum, URL identifyable to facilitate linking even between non cooperative systems.

6.1.2. Inter-Service

For the needed Inter Service communication, organization of activities and interoperation related activities, NEANIAS offers a single ticketing system through which technical partners can coordinate their activities. The scope of this ticket system is to cover:

- Interoperation protocols
- API harmonization
- Release processes
- Deployment and runtime issues
- Issue and Service request tracking within the project

The target users of this ticketing system are the NEANIAS consortium members and the tracked issues involve the interoperation and management of the services within the context of NEANIAS.

For this purpose, the ticketing system that will be used is the one that is already available at: https://ticketing.neanias.eu/projects/neanias-software

For issues involving the underlying physical or virtual infrastructures described in Section 8, the specific provider's ticketing systems should be employed. This allows to involve key persons, which may not be directly involved in NEANIAS, in the issue resolution.

6.1.3. Intra-Service

For Intra Service incident reporting, resolution tracking, planning and management, the Service Providers are free to utilize existing organization ticketing systems. The scope of these ticketing systems is to cover:

- Lower level development activities
- Backlog curation, prioritization and planning of upcoming releases
- Tracking of actively developed features
- Bug tracking and resolutions

These activities are bounded and limited in scope to intra service topics.

It is also possible for a service provider to utilize the NEANIAS provided issue tracking system that is already available at: https://ticketing.neanias.eu/projects/neanias-software

Service Providers that use the existing ticketing solution offered by NEANIAS can on demand request the creation of sub-projects within the ticketing system to gain additional encapsulation of their activities.

6.2. Processes and Flows

The process of tracking the tickets through the respective tools and namespaces, is governed by specific rules to facilitate proper, timely and effective management. Additionally, the feedback received through these tickets can be utilized in a series of flows and processes pertinent to the Service Management level that NEANIAS services target to achieve.

6.2.1. Incidents and Service Requests

The purpose of this process is to restore agreed service operation within the agreed time after the occurrence of an incident, as well as to respond to user service requests [2]. The input to

this process is in fact the ticketing system and specifically the incidents reported by users or service providers and the service requests raised by users.

The process and flow include:

- Management of incidents – Record, classify, prioritize, escalate, resolve, close, review an incident or service request
- Maintain a step by step workflow for approaching, handling and investigating these requests
- Efficiently and consistently handle well-known or recurring incidents and standardized service requests

6.2.2. Problem Management

The ticketing system is an invaluable tool when it comes to Problem Management [2] as through it, it is possible to identify the need and oversee the investigation of the root causes of incidents to avoid future recurrence by solving the underlying problem, or to ensure workarounds.

The ticketing system can be used to:

- Manage the problem – Identify and record, classify, prioritize, escalate, resolve and close the problem
- Drive and use a Known Error Database (KEDB) – Add an error and the respective workaround to the KEDB, update or remove a known error

The Know Error Database (KEDB) does not need to be maintained within the ticketing system, and in fact for NEANIAS, this is maintained in the respective section available through the published Service documentation. Still, the ticketing system is used as one of the possible ways to feed and externalize the information maintained there.

6.2.3. Change Management

The input received through the ticketing system drives changes in NEANIAS services, along with the planned feature roadmap. Additionally, the ticketing system itself, is the means by which proper Change Management can be supervised. The purpose of the Change Management [2] process is to ensure that any changes in the offered services is planned, approved, implemented and reviewed in a control manner.

Having as input any existing requests for changes as well as the information on planned releases and deployment, this process, through the usage of the respective ticketing system will:

- Manage the change request – Record, Classify, Evaluate, Approve, Implement, Review the progress of the change request
- Coordinate and provide feedback to the release planning process to schedule the changes

6.2.4. Continual Service Improvement

The input from the feedback received through the available ticketing and Help Desk tools will be used to identify, plan, implement and review improvements to Services [2].

Although the ticketing systems are only one source of input for this activity, it is essential to assist in:

- Identifying nonconformities and deficiencies in the effectiveness of the Service Management process
- Identify deficiencies in the performance of services, supporting service components and finally translating these in opportunities for improvements

Having the input collected both through the ticketing system as well as other project and service level processes, it is possible to:

- Identify and record opportunities and suggestions for improvements
- Prioritize these suggestions
- Evaluate and approve the suggestions

The output of this process is to submit suggested improvements as requests for changes per the respective process.

7. Monitoring and Alerting

7.1. Monitoring

NEANIAS services must provide monitoring means in order to ensure that SLAs and other KPIs are met. The recommended ways of providing service monitoring are through Prometheus exporters and Nagios NRPE plugins. The collected metrics can then be visualized through Grafana. These tools are described hereafter.

Prometheus [14] is an application for monitoring and alerting. It was developed at Soundcloud and inspired by *Borgmon*, i.e. Google's internal monitoring system. In the architecture of Prometheus, a centralized server collects time series metrics coming from different "exporters", which provide HTTP endpoints.

Nagios [15] is a popular monitoring and alerting application which targets services, servers and networks. Its monitoring capabilities can be easily expanded through plug-ins. Its architecture includes daemons distributed over different servers, in order to monitor remote systems. Moreover, Nagios can be enabled to collect metrics which can then be exported.

Both Prometheus and Nagios can feed data to Grafana [16], a software suite for visualization and alerting. Indeed, Grafana supports several pluggable data backends so that data coming from heterogeneous sources can be visualized, and allows a very good degree of flexibility and personalization in graph definitions.

NEANIAS services are expected to provide data to the monitoring system through Prometheus exporters or Nagios NRPE facilities. Some standard components, such as HAProxy, support Prometheus natively. In the other cases, the Prometheus website reports a list of exporters at [17]. Alternatively, the development of custom Nagios NRPE plugins typically requires low effort [18].

As NEANIAS services may be geographically distributed on different clouds, to e.g. implement high availability, incoming traffic firewall rules should be configured in order to allow connections from the monitoring systems. Where this is not applicable, Nagios passive check mechanisms or overlay networks will be considered.

The recommended minimal metrics to be collected for NEANIAS services include the service availability/uptime and the number of returning users along with used vs free server resources, total number of (HTTP) requests, server errors.

The monitoring facilities will be used to gather the service operation metrics to be reported to EOSC described in Section 7.2 and to implement the FitSM processes described in Section 2 and related corrective actions.

7.2. Quality Metrics

This section addresses metrics related to operational services. These generic quality metric recommendations should be adapted to meet each specific service characteristics, and to this aim the aggregation of metrics should be considered where suitable. Please note that we do not include metrics addressing software quality in this document, as these metrics will be addressed in Deliverable D7.3.

The recommended quality metrics to be employed include the following:

- Metrics related to service operation, used by the SLM, SACM, ISRM, ISM and ISRM FitSM processes (§ Section 2) and by the Alerting System (§ Section 7.3):
  - SLA (Service Level Agreement)
  - OLA (Operational Level Agreement)
  - Returning users
  - Number of (HTTP) requests
  - Latency of HTTP requests
  - Network reachability
  - Network latency
  - Number of errors
  - Number of pending security updates
  - Incidents
- Service metrics to be reported towards EOSC (through the available EOSC interfaces and facilities) and used by the SLM, SACM and ISRM FitSM processes:
  - Service requests
  - Service users
  - Service usage
  - Service capacity
  - Service coverage
  - Service cost
  - Service availability
  - Service reliability
  - Service serviceability/durability
- Metrics related to user support, to be used by the CRM, RDM and ISRM FitSM processes:
  - Turnaround time for issues initiated by external users
  - Turnaround time for issues initiated by NEANIAS internal users

7.3. Alerting

A NEANIAS status dynamic page will be set up, to update NEANIAS services users on the status of the infrastructure and applications and to announce maintenance operations.

There are two potential EOSC-related services to be utilized for this service implementation:

- **ARGO Monitoring**

  ARGO is a flexible and scalable framework for monitoring status, availability and reliability

  ARGO provides monitoring of services, visualization of their status, dashboard interfacing, notification system and generation of availability and reliability reports. The dashboard design enables easy access and visualization of data for end-users. Third parties can gather monitoring data from the system through a complete API. A central deployment of the ARGO monitoring engine can serve a large infrastructure reducing the maintenance costs.

Service endpoint: http://argo.egi.eu

Service information page: https://wiki.egi.eu/wiki/ARGO

- **ARGO Messaging Service**

  AMS enables reliable asynchronous messaging for the EOSC-hub infrastructure

  AMS provides a scalable HTTP Messaging Service with:

  - An HTTP API for client access
  - Transparent scalability & high availability
  - Access controls implemented at the API layer
  - Multi-tenant support
  - Instrumentation at the API layer

  Service endpoint: http://argoeu.github.io

  Service information page: https://wiki.egi.eu/wiki/Message_brokers

Integration of these system with EOSC alerting services will be evaluated.

8. Infrastructure Access

This section describes the physical and virtual infrastructures which are planned to be employed for NEANIAS services development and delivery.

8.1. GARR Cloud

GARR, the Italian National Research and Education Network, operates an IaaS facility for the research community [19].

The GARR Cloud IaaS is based on OpenStack and spans across three separate geographical regions (Catania, Palermo, Napoli) interconnected through high capacity fiber optic links.

The full ICT infrastructure includes also the sites of Bari and Cosenza for a total of 4224 Cores and about 10 PB of raw disk storage organized in 11 racks distributed among the sites (3 each in Bari, Catania and Palermo; 1 each in Cosenza and Napoli).

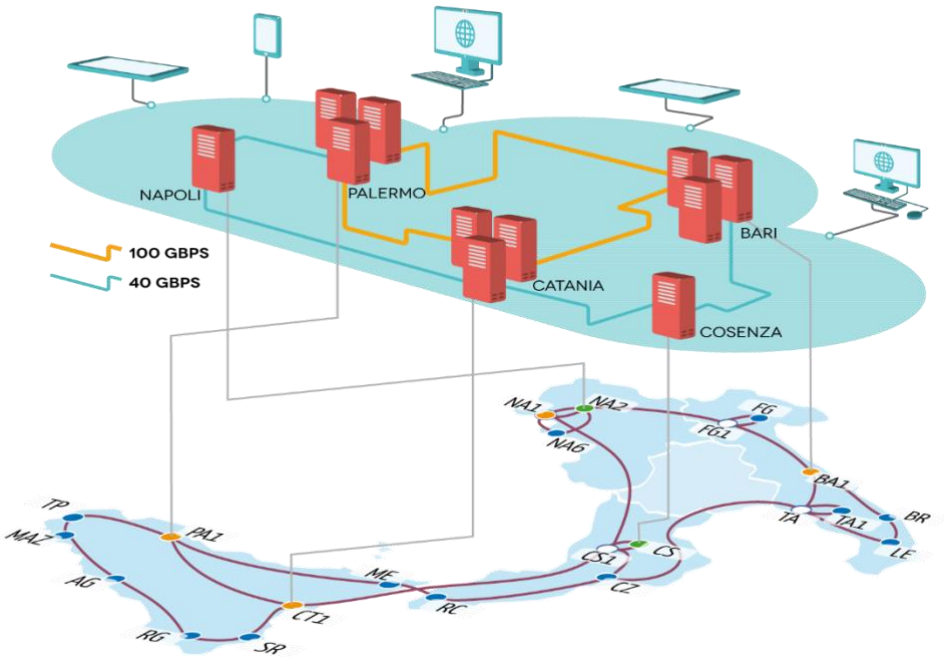

Figure 3 – GARR Cloud Infrastructure

Users can login to the OpenStack dashboard using federated identities systems (IDEM/eduGAIN), thus employing credentials from their home institutions.

Along with the OpenStack deployment, GARR operates a (PaaS) Kubernetes service featuring GPUs, which can be accessed with the same credentials as OpenStack.

For NEANIAS, the following OpenStack projects have been created:

- NEANIAS-atmospheric: for the atmospheric sector services
- NEANIAS-underwater: for the underwater sector services
- NEANIAS-space: for the space sector services
- NEANIAS-core: for core services (WP6)
- NEANIAS-delivery: for the software delivery facilities (WP7)

8.1.1. GARR Cloud Access Procedure

To access the GARR Cloud infrastructure the following steps should be followed by NEANIAS participants:

1. Edit the file NEANIAS Infrastructure Accounts.xlsx in the WP7-Delivery/Infrastructure folder adding name and surname, e-mail address and the OpenStack projects to which users need to be added;
2. Browse to https://signup.cloud.garr.it and register to the GARR Cloud. The preferred registration method is IDEM/eduGAIN. Only if the user's institution does not support this type of authentication the Google based authentication or the Form based authentication should be used;
3. Wait for registration approval, and/or notify GARR members through the NEANIAS communication channels (i.e. Teams);

After the approval, users should be able to login into https://dashboard.cloud.garr.it with the chosen authentication method, to select a project and region on the upper part of the page and create new VMs as described at https://cloud.garr.it/compute/quick-vm/.

The *garr-na* (Napoli) region deployment has a more "experimental" OpenStack setup, thus is considered less stable and more suitable for tests than production deployments.

To access Kubernetes, users can follow the instructions at: https://cloud.garr.it/containers/. Please note that GPU access is subject to availability.

The GARR Cloud documentation is available at https://cloud.garr.it/. For GARR Cloud related issues, NEANIAS service providers can receive support via email (cloud-support@garr.it).

8.2. MEEO Cloud

The MEEO Data Center facility is an infrastructure created to support the high computing demands of MEEO geospatial web platforms to manage geospatial data and services. Initially designed for Earth Observation data services, it has been extended to provide computational resources and hosting to a wide range of geospatial datasets. The facility gives academic, government and commercial users access to internally hosted climate and EO Data alongside a wide range of associated Services for data processing, visualization and analysis.

The fundamental characteristics of this data center facility are:

- High performance computing facilities collocated with private storage areas and large volume community satellite data sets;
- A scalable model for developing commercial and academic services and applications;
- Virtual environment for supporting any kind of virtualization needs;

- A key focus on the development and use of data integrity tools to clearly articulate the accuracy and provenance of (meta-)data, enabling users to be able to rely on the data/information;

The MEEO Data Center is colocated in two sites, at the University of Ferrara and at the Ferrara Datacenter, interconnected to the 1Gbps high-speed backbone GARR network.

An OpenNebula instance manages the MEEO Data Center virtualization resources, currently crossing +1PByte of online storage, 3.5PB off-line storage, +1000 cores, +4.5TB of RAM.

8.3. CITE Cloud

CITE's infrastructure is located in two locations: (a) Kessariani, Attiki Greece and (b) Sparta, Laconia, Greece, where the main Data Center is located (on premises).

That infrastructure is used for CITE's day-to-day operation, for providing services to third parties (including Microsoft Services through Microsoft Services Provider License Agreement) and mainly for supporting CITE's R&D.

The infrastructure is mainly based on Microsoft Hypervisor and is hosting both Microsoft Windows VMs and Linux VMs.

CITE will have at NEANIAS's disposal up to 40 vCPUs, 512GB RAM, 20TB storage and allocate 50MBps total network, offered from CITE's on premises Computer Room in Sparta's branch office. Those resources will be offered to support showcasing and operation for the project duration, as well as for pre-production and staging needs.

8.4. MTA Cloud

MTA Cloud (https://cloud.mta.hu) is a community cloud for researchers and scientists belonging to any of the research institutes of the Hungarian Academy of Sciences.

The infrastructure is consisting of two datacenters. One datacenter, operated by the Institute for Computer Science and Control (SZTAKI), is located near the centre of the city, in the vicinity of the Hungarian Internet Exchange Point (BIX). The second datacenter, operated by the Wigner Research Centre for Physics, is located at a secure facility on Budapest hillside, which has been recently repurposed from providing Tier-0 services for CERN WLCG. A joint undertaking of the two datacenters, focusing on supporting the Hungarian research community, was sponsored by the Hungarian Academy of Sciences (MTA) since 2016 until August of 2019. Since September 2019 the infrastructure is sponsored by ELKH (Eötvös Lóránd Research Network) that plans to support a significant further development of the infrastructure increasing the overall capacity 4 times bigger than the current capacity. The datacenters work as an alliance and are facilitating similar infrastructures including several shared integration components, but have a distinct budget, as well as separate design and operation teams.

The joint IaaS/PaaS service, currently known as "MTA Cloud", has gained serious momentum amongst researchers, counting more than 110 projects approved by the project committee in the last three years. The effort is also catalyzed by a series of in-depth workshops, specialized service patterns supporting various scientific scenarios, a shared web portal and common authentication framework. Unique value of the service lies in close cooperation of scientific

and technical personnel in a wide range of projects, resulting in a continuously evolving knowledge base for researchers and IT professionals alike.

Both datacenters are based on commodity hardware, operating on the OpenStack IaaS platform with CEPH based storage. Current capacity is 1368 vCPUs, 3.25 TB RAM, 527 TB HDD storage in a 10Gbps networking environment. Public procurement tenders are being initiated for 2020 with an approximate budget of 3.000.000 EUR, these will substantially improve the service, resulting in overall capacity of ~3500 vCPUs, ~12 TB RAM, ~150 TB SSD storage and ~1400 TB HDD storage in a 100Gbps internal and external networking environment. With current research focus of AI specific applications, development goals also include 2048 GPGPU cores with a total tensor capacity of ~7 PFLOPS. Resulting total capacity will be distributed between the two datacenters in an even manner.

New capacities mentioned above are planned to be available for researchers by the end of 2020.

Since MTA Cloud is a community cloud, it is planned to be only used by SZTAKI for testing and development purposes.

8.5. NKUA Cloud

UoA's team that participates in the project maintains a cluster of servers of more than 3 TB of RAM and 350 cores followed by 100 TB of high availability storage.

This cluster is used by the team's researchers and scientists and is contributing resources to many projects in which the team is participating.

This cluster is colocated at the Data Center of the Department of Informatics & Telecommunications of the National and Kapodistrian University of Athens, at Panepistimiopolis, Ilisia, Athens, Greece. The Data Center is connected to the internet through a 10 Gbps line as well as a 1Gbps backup internet line.

For operational purposes of NEANIAS, NKUA will provide up to ~40vCPUs, 256GB vRAM and 2 TB of storage, from the above cluster, with 1Gbps internal network connections. The infrastructure is based on both XEN hypervisor and Microsoft Hypervisor. The VMs are provisioned by the team's administrators, after examining requests and intended usage. If needed, NKUA will accommodate additional requests from GRNet’s Oceanos cloud (LOS).

8.6. ALTEC Development Cloud Platform

ALTEC has the availability of a platform based on a modular infrastructure and software defined technologies that, efficiently, combines processing, storage and connectivity resources. The ALTEC Development Cloud Platform will be used for the work offered to set up and configure the requested “Development” and “Integration” Infrastructure environment for WP4 Space Services and part of WP6 Core Services. The platform usage costs are indirectly charged to the Contract. The platform is easily upgradable, adding processing, storage and connectivity modules but no modifications are expected for the project purpose as the available resources are sufficient to support the execution of the proposed activities.

The current platform infrastructure is assembled using HPE SYNERGY components, allocated to three frames configured in high availability mode, and it has 18 nodes and approximately

110 TB of usable space (270 TB raw). Platform external connectivity is provided through 4 x 40 Gbps fiber connections attested on the ALTEC backbone (100 Gbps).

VmWare, OpenStack, CEPH, Cloudera and Kubernetes are the software environments available to set up the project infrastructure layer. The platform implements the cloud hybrid model therefore, in case of necessity, the project software environment could migrate to public cloud platform.

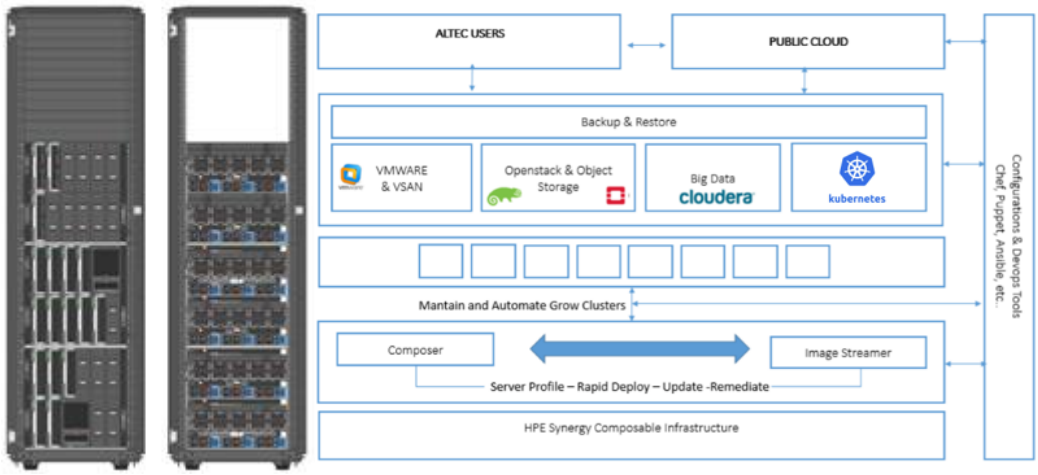

Figure 4 - ALTEC Development Cloud Platform : HW and SW Environments

ALTEC infrastructure is located inside CMFS Center in Turin connected with ALTEC's network (backbone is at 100 Gbps and managed by CISCO ACI Network Infrastructure) through two 40 Gbps connections.

Internet access is guaranteed by two independent ISPs. One is GARR, providing internet access al 1 Gbps (300 Mbps are guarantee) and the second one is COLT at 300 Mbps.

Internet access and VPN connections are controlled by a redundant firewall based on the PfSense SW firewall (in Active-Standby configuration).

9. General Recommendations and Remarks

9.1. Cloud Infrastructure Reliability

Compared to on-premises infrastructures, deploying services on the cloud requires a paradigm shift: indeed, cloud datacenters are usually built using cheap hardware, with the idea of quickly replacing failed components. Thus, applications in the cloud should not rely on the availability of the underlying infrastructure, and should implement high availability (HA) at the application layer. To this aim, several service providers are leveraging chaos engineering practices [23] to ensure that services are designed to be available despite failures in the underlying systems.

As failures may as well occur in the infrastructures hosting the NEANIAS services, the recommendation is to adopt high availability service design principles and to leverage on deployments on different geographical regions.

9.2. Infrastructure as Code

The main idea of the Infrastructure as Code (IaC) paradigm is to use textual infrastructure descriptions, which can be then employed to execute automatic service deployments, and which can be managed through revision control systems.

This allows to automate the deployments and have an always up to date documentation of the processes.

Thus, the recommendation is to employ IaC oriented tools, such as Dockerfiles, Kubernetes resource files and Ansible playbooks, while the use of binary images or artifacts is not recommended.

9.3. Service Registration and Configuration

According to the Service Oriented Architecture (SOA) [21], the process by which a consumer locates a required service often requires that some means of discovery mechanism is employed. Within the NEANIAS ecosystem, the Service Catalog will have the role of describing the services following the EOSC standards for service description, registration and configuration and facilitate the discoverability of the service offering both to end users as well as to other services.

This mechanism though includes several steps, possibly offline communications, quality checks and other processes. After a service has been successfully catalogued, running instances of the service need to be registered, differentiated based on their capacity, quality of service, configuration and a number of other service specific characteristics. This registration does not necessarily match the purposes of the Service Catalog as a “customer-facing list of all live services offered along with relevant information about these services” [2].

Regardless of whether the service running instance registry is maintained under the same enabling service of the Service Catalog or as a different stand-alone service, the ability to dynamically discover and utilize instances of a service is paramount to the proper decoupling of different services and can become a key enabling element when servicing specific use cases with targeted requirements.

Equivalently to the dynamic discovery of services, the ability to dynamically retrieve service configuration and parametrize the operation of each service is a key enabler to some automatic service provisioning use cases. For this characteristic to become available, related configuration needs to be centrally controlled and provided to bootstrapped services at the appropriate time.

9.4. Service Versioning and Upgrading

During the lifetime of the NEANIAS services, upgrades of both the service components, the exposed API and possibly the underlying data model may be required. These changes need to be communicated to the service clients as well as ensure the continuity of the offered service. To this end, some versioning scheme needs to be employed to differentiate between different service level offerings and respective functionality.

The Semantic Version scheme [20] is proposed for its simplicity and wide adaptation. Regardless of whether versioning is employed both at the service level or at the underlying data model, it is required that it is used at all interfacing components of the service in a consistent fashion. In a nutshell, given a version number MAJOR.MINOR.PATCH, increment the:

- MAJOR version when you make incompatible API changes,
- MINOR version when you add functionality in a backwards compatible manner, and
- PATCH version when you make backwards compatible bug fixes.

Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

When upgrading a service and affecting its respective version number, the process must be well defined and reproducible in an automated fashion to the highest possible degree. This includes all possible changes, including data model changes, service components, configuration, hosting environment, etc.

9.5. Testing & Development Infrastructures

The development cycle of a service will require that it passes various stages and respective infrastructures that support the service provision lifecycle. These include:

- Development – The service is actively developed
- Testing / Quality Assurance – The service is tested to ensure proper operation and integration
- Staging – Pre-production environment where the service is used, tested and evaluated in “close to production” environment
- Production – Service fully operational and service clients

Throughout these stages, depending on the nature and purpose of the service, a hosting environment will be needed to cover the execution of the service itself, underpinning dependencies, computing and storage requirements, etc. This environment and corresponding setup will differ from service to service.

Service providers need to formalize their requirements per service for each of the key aspects that are affected by the hosting environment. This includes both what is required by them as

well as if and how they can, on demand, service client requests within each of these environments.

Some, non-exhaustive, options include:

- Provide mock service for integration purposes
- Provide sandbox environments within existing installations of the service
- Use production or otherwise shared instances of the service
- For data dependent services, provide synthetic datasets

Concerning infrastructural requirements, infrastructure providers need to be able to provide isolated pools of resources to cover these service requirements.

With respect to resource utilization and accounting, non-production instances that consume such data and processing resources from other services, need to do so under a common agreement and possibly throttle the resource consumption based on service provider constraints.

10. Work Plan

The workplan for delivering all NEANIAS services, thematic and core, is depicted in Table 1 - Workplan for delivering NEANIAS services below.

The columns are the months counting from the project start date. In the release description column, “Rx” means “Release x”. The initials in the workplan cells are defined as follows:

- I (blue): Integration test release. This is a pre-release version, provided to allow consumers of the service to verify their integration with the released component.
- T (orange): Test release. This is a beta-level release of the service, to be tested from a wider audience than the developers of said service.
- P (green): This is the production-level release. At this point, the service is fully operational and accessible by all its intended audience.

Release description	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Core services R1	I	T	P																						
Underwater Services R1			I	T	P																				
Atmospheric Services R1			I	T	P																				
Space Services Release R1			I	T	P																				
Business innovation cases software release R1							I	T	P																
Core services Release R2												I	T	P											
Underwater Services R2														I	T	P									
Atmospheric Services R2														I	T	P									
Space Services Release R2														I	T	P									
Core services Release R3																			I	T	P				
Underwater Services R3																					I	T	P		
Atmospheric Services R3																					I	T	P		
Space Services R3																					I	T	P		
Business innovation cases software R2																							I	T	P

Table 1 - Workplan for delivering NEANIAS services

NEANIAS is funded by European Union under Horizon 2020 research and innovation programme via grant agreement No. 863448

11. Conclusions

This document reports a description of how the NEANIAS project is implementing the FitSM service management processes. To this aim, software implementation facilities, software documentation and software licensing guidelines are described. For running services, feedback and issue tracking tools and processes are presented as well as monitoring and alerting systems. The underlying physical infrastructures are then described. Important general recommendations are also reported. Finally, a work plan is provided for the NEANIAS service delivery activities.

NEANIAS is funded by European Union under Horizon 2020 research and innovation programme via grant agreement No. 863448

References

[1] FitSM: https://www.fitsm.eu/
[2] FitSM-0 Overview and vocabulary: https://www.fitsm.eu/download/280/
[3] FitSM-1 Requirements: https://www.fitsm.eu/download/295/
[4] FitSM-2 Objectives and Activities: https://www.fitsm.eu/download/304/
[5] Gitlab https://about.gitlab.com/
[6] Introduction to CI/CD with GitLab: https://docs.gitlab.com/ee/ci/introduction/index.html
[7] Gitlab CI/CD: https://docs.gitlab.com/ee/ci/
[8] Ansible: https://www.ansible.com/
[9] Kubernetes: https://kubernetes.io/
[10] Overview of Docker Compose: https://docs.docker.com/compose/
[11] Occopus: http://occopus.lpds.sztaki.hu/
[12] Github: https://github.com
[13] Gitlab - repository mirroring: https://docs.gitlab.com/ee/user/project/repository/repository_mirroring.html
[14] Prometheus: https://prometheus.io/
[15] Nagios: https://www.nagios.com/
[16] Grafana: https://grafana.com/
[17] Prometheus exporters and integrations: https://prometheus.io/docs/instrumenting/exporters/
[18] Nagios Plugins Development Guidelines: https://nagios-plugins.org/doc/guidelines.html
[19] GARR Cloud Platform: https://cloud.garr.it/
[20] Semantic Versioning 2.0.0: https://semver.org/spec/v2.0.0.html
[21] Reference Model for Service Oriented Architecture 1.0: https://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf
[22] The Git version control system: https://git-scm.com/
[23] The Netflix Simian Army: https://netflixtechblog.com/the-netflix-simian-army-16e57fbab116
[24] OpenStack Object Storage API: https://docs.openstack.org/api-ref/object-store/
[25] Openstack Glance documentation: https://docs.openstack.org/glance/latest/
[26] ITIL v4: https://www.axelos.com/welcome-to-itil-4
[27] FitSM - IT Service Management 'Lite': https://apmg-international.com/article/fit-lightweight-and-lean-fitsm
[28] OpenAPI Specification: https://www.openapis.org/
[29] FitSM - Simple SLA: https://www.fitsm.eu/download/357/
[30] FitSM – Corporate: https://www.fitsm.eu/download/333/
[31] Introduction to Singularity: https://sylabs.io/guides/3.5/user-guide/introduction.html
[32] Open Source Initiative approved licenses: https://opensource.org/licenses
[33] GNU General Public License v3: https://www.gnu.org/licenses/gpl-3.0.en.html

[34] Apache License, version 2.0: http://www.apache.org/licenses/LICENSE-2.0
[35] MIT License: https://opensource.org/licenses/MIT

List of acronyms

Acronym	Description
EOSC	European Open Science Cloud
NREN	National Research and Education Network
IaaS	Infrastructure as a Service
FOSS	Free / Open Source Software
PaaS	Platform as a Service
IDEM	Italian Identity Federation
eduGAIN	International Federation of National Identity Federations
IaC	Infrastructure as Code
HA	High Availability
SOA	Service Oriented Architecture
FitSM	Family of standards for lightweight IT Service Management (ITSM)
CI/CD	Continuous Integration/Continuous Delivery
SPM	Service Portfolio Management
SLM	Service Level Management
CRM	Customer Relationship Management
RDM	Research Data Management
SACM	Service Availability and Continuity Management
ISM	Information Security Management
ISRM	Incident and Service Request Management
TRL	Technology Readiness Level
HPC	High Performance Computing

Appendix 1 – Internal Questionnaire

In the context of examining the existing service landscape as well as identifying the intentions of Service Providers with respect to key aspects of the service lifetime, a questionnaire was created and circulated to help identify suitable approaches for NEANIAS service delivery methodologies. A subset of the questionnaire is presented here and the respective feedback was used as input for this deliverable.

#	Service Details
1	The name of the service
	The Service Provider
2	Lead Provider Name
3	Collaborating Providers
4	Webpage
	The Service Owner/ Main Contact
5	Name
6	Email
7	Organisation
	The thematic the service falls under
8	Scientific Domain
9	Scientific Subdomain
10	Category
11	Subcategory
12	Tags
	Scope
13	What is the nature of Service you are providing (more than one can be mentioned) ? [Infrastructure (compute / store / …), Web Service, CLI, Component (library), Web app, Widget, Other]
14	Description
15	Target Users
	Location Information
16	Geographical Availability
17	Language
	Documentation
18	Under what license are you providing your service?
19	Is there end user documentation available for the usage of your service?
20	Is there administration / system documentation available for the maintenance of your service?
21	Is there deployment / installation documentation available for the deployment and upgrading of your service?
22	Are there supplementary material like tutorials / webinars available for your service?
	Testing / QA

23	Do you perform static code analysis on your service codebase?
24	Do you follow specific code syntax rules?
25	Do you have a target test coverage percentage? If yes, what is it? (Includes any type of tests)
26	Do you perform Unit Testing on your service codebase?
27	Are the Unit Tests integrated to the build process of your service?
28	Do you perform Integration Testing on your service?
29	To what extend are your Integration Test automated?
30	During testing, for your underpinning dependencies do you use: - Mock Services - Sandboxes provided by the provider - Production Instances - Other
31	Do you perform User Interface tests?
32	To what extent are your User Interface tests automated?
33	Do you have documented user acceptance test scenarios?
34	Do you execute user acceptance tests ?
35	At the time of defining development tasks / service features, do you define acceptance criteria including test cases?
36	Do you provide sandbox deployment for services dependign on your service to conduct development and testing ?
	Building & Packaging
37	Do you use a central build system?
38	Do you employ Continuous Integration processes? If yes, which system do you use?
39	Do you employ Continuous Deployment processes? If yes, which system do you use?
40	In which form is you service packaged & distributed?Indicate explicitly if docker containers, war/jar or other well known packaging is supported (potentially more than one)
41	Do you use some automation means of provisioning, configuring, deploying your packaged software (eg Ansible, Saltstack, Pupper, Chef)?
42	Which host OS does your service requires?
43	Does you service requires pre-existing local dependencies that it does not bring along with it?
44	Does your service require manual configuration on deployment?
45	Are there any additional restrictions with respect to the deployment of your service?
46	Is your services running / could run within a managed workload / orchestration environment (Occopus / Kubernetes / Docker Swarm / Terraform / Cloudify / …)?
	Release & Code Management
47	What versioning policy do you use for your service?

48	How does you service handle backward compatibility (multiple may apply)? - Across minor releases only (implied semantic versioning) - Through parallel deployments of different API versions - Other - Not handled
49	Is your release plan governed by restrictions / planning outside the NEANIAS domain?
50	What are your service's (known) dependencies to other NEANIAS services?
51	Dependencies information - Required Services
52	Dependencies information - Related Services
53	Dependencies information - Related Platforms
54	Do you maintain a backlog of development tasks?
55	Do you organize your Service evolution through a ticketing system?
56	How do you handle production incident requests / blocking issues?
57	What is the release cycle for your service?
58	Is your service non-FOSS? Can it be freely distributed?
59	Do you maintain an internal/restricted repository for your service codebase?
60	Can you provide access to your codebase for the consortium? What kind of restrictions would apply?
61	Will you require some NEANIAS partner to provide controlled access repository for your service codebase?
62	Would you make your service codebase available under a NEANIAS github organization?
	Monitoring
63	How do you define scheduled downtime for your service (eg all announced maintenance donwtime with at least 5 days notice)?
64	How do you define unscheduled downtime for your service (eg all maintenance donwtime with less than 5 days notice / interruption of service)?
65	What are the mechanisms, rules and policies to unforeseen downtime or degraded service?
66	Do you define an availability target for your service?
67	How do you measure quality of service for your service?
68	Does your service offer health check endpoints for monitoring while in operation?
69	What kind of mechanisms for monitoring do you employ, if any?
70	Webpage with monitoring information about this service
71	Webpage with information about planned maintenance windows for this service
	Operation & Policies
72	Do you provide an SLA with your service?
73	Does your service operate under some EULA?
74	Do you offer your service under specific Terms of Services?
75	Webpage with Terms Of Use
76	Webpage with Privacy Policy
77	Do you track or maintain data that fall under the GDPR restrictions?

78	Does infrastructure or datasources affect service licensing and how?
79	Is your service being used by end users or/and by other NEANIAS's services or/and 3rd party services?
80	How do you announce downtime for your service (eg mail, bullting board, etc)?
81	What is the usual announcement period prior to scheduled downtime?
82	What is the usual announcement period prior to downtime because of unforeseen issues?
83	What is the typical downtime duration?
84	Does your service require frequent / documented administration / maintenance tasks?
85	Do you require access to the service host to perform administration / maintenance tasks?
86	How do you restrict administration tasks to authorized usage?
87	Does you service offer relocation mechanisms?
	Data
88	Is your service producing new datasets?
89	How is the target user of the datasets your service is producing?
90	Under what usage / license restrictions are the data you are producing?
91	Does your service have strong proximity requirements with respect to data?
92	How does data storage location affect the behavior of your service? Where would the data related to the service be?
93	What is your service compute and storage requirements (depending on usage)?
94	Which are the datasources consumed by your service and are they provided by you/other partner/EOSC/third parties?
	HelpDesk
95	Do you have a helpdesk for your service users?
96	Helpdesk Webpage
97	Helpdesk Email
98	User Manual
99	Training Information
100	Would have your HelpDesk under a common NEANIAS hosted one?
101	What is the average response time for a user request?
102	Do you maintain a Known Issues database to record probglems and related workarounds?
	Infrastructure
103	Does your service require NEANIAS to provide a testing infrastructure?
104	Does your service require NEANIAS to provide a QA Infrastructure?
105	Have you identified critical factors that jeopardize your Service's SLA?
106	Do you track and monitor your service KPI with respect to target SLA?
	Service Management
107	Are you / your organization familiar with any Service Management Systems?
108	Do you employ some Service Management process for your service?

109	Do you have clear role distinctions within your organization with respect to the management of your Service? - Service Management System Owner / Manager - Service Owner - Process Owner / Manager - Case Owner - …
110	Do you have clear role assignments and responsibilities within your organization with respect to the management of your Service? - Described with a RACI matrix for example?
111	Do you review the processes around the management of your service in regular intervals?
112	Do you have a single and consistent location and format for your service management documentation? - Eg document management system, wiki, versioning system etc
113	Is your service management policies aligned in scope with the target customer requirements and available resources?
114	Have you defined the target level of maturity for service management to be achieved? Eg: - Level 0: Unaware/ Non-existent. There is no awareness of the task at hand. / The required output does not exist. - Level 1: Ad-hoc / Initial There is awareness of a task but it is uncontrolled. / Some relevant output exists but with core elements missing. - Level 2: Repeatable / Partial Tasks are repeatable but not formally defined. / Outputs are only partially complete. - Level 3: Defined / Complete Tasks are well defined and outputs are complete, both are connected to documented responsibilities
115	What is the gap between current and targeted maturity level for your service management plan?
116	What tools to you use to track the implementation of your Service Management Plan?
117	How do you track the effectiveness and efficiency of your Service Management System?
118	How do you track the effectiveness and efficiency of your Service Management Plan?
119	Do you have a service portfolio escribing the servicers offered?
120	Do you maintain an up to date map between your services underpinning services and their owners / responsible parties?
121	Do you have a capacity plan in place to meet requested capacity and performance requirements?
122	How do you classify the assets your service operates on based on their sensitivity / criticality?
123	Do you have a process to define and assess information security risks?
124	How do you define the most important information security assets?
125	Do you maintain a service user database and act on it to ensure user-provider relations?
126	Do you have a process to respond to service user complaints?
127	Do you have defined means to measure user satisfaction?
128	Do you have a classification policy to distinguish between incident report severity?
129	Do you have a standardized procedure to escalate incident and service requests?

130	Do you have a ticketing system in place with workflows to support the recording and handling of incidents and service requests? - Including classification, prioritization, escalations, closure,…
131	Do you have a process of documenting and evaluating service change requests?
132	Would you be willing to invest in getting all of the above and more?

Table 2 - Service Landscape Questionnaire

Some of the findings from the feedback received from the service providers that answered the questionnaire are summarized below:

- License – Most of the service providers will offer the respective NEANIAS services under a permissive license. Some services will require mixed licenses including both free and restricted components, while a smaller portion of services may require more restrictive licenses to some of its components.
- Documentation – The majority of the service providers intend to provide all needed user documentation and support material. The documentation that will be made available for administration, system and deployment purposes will be provided as required to facilitate the relevant needs.
- Code Quality & Testing – Code management tools and quality approaches such as static code analysis, consistent style checks, and more are not equally followed by all service providers. Still the majority of them are either already utilizing, or plan to utilize more testing methodologies at the unit, integration and user level, in a mixed level of automation.
- Continuous integration – Most of the service providers do not utilize consistently a CI/CD pipeline although most are willing to integrate such approaches in their development, testing and release lifecycle
- Packaging – Most of the service providers aim to provide their services as Docker container while other technologies may also be utilized such as Singularity or Virtual Machines
- Host Operating System – Most services seem to only require a run time environment that can support the respective containerization approach. Wherever specific Operating System is requested, it is mostly Linux or Unix based with few exceptions
- Version Compatibility - Most of the Service Providers do not have specific plans to handle backward client compatibility and will require that the latest versions of their services will be used with any API changes across versions
- Ticketing – Most of the Service Providers already use some ticketing system to handle service evolution. Those that do not already do so, are expressing their intention to utilize such a tool as provided by NEANIAS
- Source Code Repository - Most of the Service Providers already use some source control management system to track their code base evolution. Those that do not already do so, are expressing their intention to utilize such a tool as provided by NEANIAS

- Public code repository – Most of the Service Providers intend to provide their service code base as FOSS through a NEANIAS hosted public repository. Exceptions to that include proprietary software components and other externally hosted components
- Monitoring – Although some of the existing Services do not have an already formulated approach to consistent monitoring solutions, it is evident that the Service Providers are willing to comply to a uniform solution that NEANIAS will put forward, to the extend that this will not be over-intrusive
- Service Level Agreements – Most of the Service Providers do not yet have formulated SLAs through which they offer their services but are in the process of defining such agreements for their users
- GDPR considerations – Few services handle personal and or sensitive user data and are aware of the restrictions and obligations that these impose
- Data Producer – Most of the thematic services fall under the category of data producers and depending on the dataset they consume, restrictions on the licensing of their product may arise
- Data Consumers – Most of the data consuming services note that they have a dependency on the location of the data which they require to have in close proximity
- HelpDesk – Although most of the Service Providers do not currently have a full Help Desk solution, the willingness to utilize a NEANIAS offered approach for it is expressed
- Infrastructure – Most of the services will require that relevant development, staging and production environments are made available to facilitate the service delivery and desired quality standards
- Service Management – Service Providers were asked to choose the desired target level of maturity for the service management processes they were willing to invest on among the following ones:
    - Level 0: Unaware/ Non-existent. There is no awareness of the task at hand. / The required output does not exist
    - Level 1: Ad-hoc / Initial There is awareness of a task but it is uncontrolled. / Some relevant output exists but with core elements missing.
    - Level 2: Repeatable / Partial Tasks are repeatable but not formally defined. / Outputs are only partially complete.
    - Level 3: Defined / Complete Tasks are well defined and outputs are complete, both are connected to documented responsibilities

  Among the responses, a varying level of desired conformance was expressed. Most of the Service Providers are currently investing on some Service Management processes as best fits their internal needs. Although the desire to formalize the processes is expressed, service providers express different expectations on the end target