H2020-EINFRA-2017

EINFRA-21-2017 - Platform-driven e-infrastructure innovation

DARE [777413] "Delivering Agile Research Excellence on European e-Infrastructures"

D5.2 Platform Infrastructure, Usage & Deployment II

Project Reference No	777413 — DARE — H2020-EINFRA-2017 / EINFRA-21-2017
Deliverable	D5.2 Platform Infrastructure, Usage & Deployment II
Work package	WP5: Platform Operation and Maintenance
Tasks involved	T5.1: Provision of Relevant Cloud Infrastructure T5.2: Provision of Pre-release Testbeds T5.3: Deployment Strategy and Platform Operation
Type	R: Document, report
Dissemination Level	PU = Public
Due Date	31/12/2020
Submission Date	30/12/2020
Status	Draft
Editor(s)	M. Roth (SCAI), H. Schwichtenberg (SCAI), A. Gemünd (SCAI)
Contributor(s)	S. Sachtouris (GRNET)
Reviewer(s)	F. Magnoni (INGV)
Document description	This deliverable represents an update of Deliverable D5.1. It presents an overview of the platform design focusing on infrastructure components, depicts the deployment strategy employed throughout the project and provides related usage information. Moreover, it

	specifies the underlying computational resources mobilized by GRNET and SCAI to provide development and operation of the DARE platform.

Document Revision History

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v1	14/09/2020	Initial Structure	M. Roth (SCAI)
v2	14/10/2020	First Draft	M. Roth (SCAI)
v3	12/11/2020	Revision	H. Schwichtenberg (SCAI)
v4	09/12/2020	Revision	A. Gemünd (SCAI)
v5	09/12/2020	Chapter 3.2	S. Sachtouris (GRNET)
v6	10/12/2020	Final version	M. Roth (SCAI)
v7	10-11/12/2020	Internal review	F. Magnoni (INGV)

Executive Summary

The present document reports on the work performed in the scope of WP5, which was responsible for the provision and management of the cloud environments made available for the DARE platform. This included the deployment, configuration and operation of required infrastructure, system software and DARE components.

This deliverable gives an overview over the platform design focusing on infrastructure components, reports on the deployment strategy, presents the deployment schedule and provides related usage information. Moreover, it specifies the underlying computational resources mobilized by GRNET and SCAI.

Table of Contents

List of Figures

List of Tables

List of Terms and Abbreviations

Abbreviation	Definition
AAI	Authentication and Authorization Infrastructure
API	Application Programming Interface
AWS	Amazon Web Services
CPU	Central Processing Unit
EGI	European Grid Infrastructure
EOSC	European Open Science Cloud
GRNET	Greek National Infrastructures for Research and Technology
HPC	High Performance Computing
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
MPI	Message Passing Interface
MS	Milestone
RBAC	Role-Based Access Control
SCAI	Scientific Computing and Algorithms Institute (Fraunhofer)
SSO	Single Sign-On
WP	Work Package

1 Introduction

One of the central goals of the DARE project was the development of a cloud-ready platform, supporting research developers in utilizing the diverse landscape of e-infrastructures and handling the complexity of diverse data sources, complex computations and computational contexts for their scientific workflows. WP5 - Platform Operation and Maintenance, was responsible for providing and managing the operational and testing cloud environments for the DARE platform, which included the deployment, configuration and operation of required infrastructure, system software and DARE components. Moreover, WP5 developed a packaged platform solution that is easily deployable e.g. on public clouds like European Open Science Cloud (EOSC). Also the provision of complementary guidelines and usage information of the platform was part of WP5.

1.1 Purpose and Scope

The purpose of this document is to report on the work performed in T5.1: Provision of Relevant Cloud Infrastructure, T5.2: Provision of Pre-release Testbeds and T5.3: Deployment Strategy and Platform Operation. It gives an overview over the platform design focusing on infrastructure components and specifies the underlying infrastructures (testbed and productive environment). Connecting to D5.1, the applied deployment strategy is outlined and the updated deployment schedule of all platform releases is presented. Finally, it provides related usage information.

1.2 Approach and relation with other Work Packages and Deliverables

While this document focuses on the description of infrastructure aspects, D5.4- Platform Operation and Maintenance II, produced in the scope of T5.4, provides the related installation, deployment and operation documents for the DARE platform.
Altogether, the infrastructure specification of the DARE platform developed in WP5 is in alignment with the architecture developed in WP2 and motivated by the requirements and the feedback provided by the use cases in WP6 and WP7.
In cooperation with WP3 and WP4, WP5 performed the containerisation of the DARE components and offered support with the integration and testing of software assets, tools and services developed in the other WPs.

1.3 Methodology and Structure of the Deliverable

Initially, in chapter 2 we shortly describe the general strategy followed to provide the infrastructural framework for the development and provision of the DARE platform. In chapter 3, we summarize and explain the relevant infrastructure components, specify the underlying hardware of the production platform as well as the testbed environment and briefly describe the relation to EOSC. In chapter 4, we outline the deployment strategy already introduced in D5.1 and align it to the latest status. Finally, in chapter 5 we provide further information regarding the three main usage purposes of the DARE platform during the project phase, namely development, testing and training.

2 Platform Infrastructure Development

The cloud-based DARE platform was developed as a user-friendly, easily deployable framework allowing to make use of available and upcoming EOSC services. However, at the beginning of the DARE project, the implementation status of the usable EOSC services was still unclear. Additionally, the project needed a directly usable infrastructure for the development and prototyping of services and their application for the use case scenarios. Therefore, the partners decided to adopt the following strategy:

In a first step, WP5 focused on providing and maintaining a pre-release infrastructure (testbed) which included the software development environment for the continuous integration and testing of the DARE software products. The aim was to support the agile software development and containerization of the DARE services right from the beginning of the project and to automate a continuous delivery. This testbed environment was provided on cloud resources at GRNET and was equipped with the same prerequisites, which were also intended to be built in the production environment.

In a second step, WP5 set up this production environment, by means of a deployed, integrated and operational instance of the DARE platform, which is provided on SCAI's cloud resources provisioned in the scope of the EGI Federated Cloud (part of the EOSC). This platform already comprises all necessary components for the use cases developed in WP6 (seismology) and WP7 (climate) and already includes a configured Authentication and Authorization (AA) solution for the related research communities. It is a demonstrator, displaying functionality and usability of the developed solution.

Additionally to this operational instance of DARE platform, partners decided also to provide a packaged version of the platform, easily deployable on cloud resources (e.g., EOSC, AWS). In this way, also users from other research communities are equipped with a basis development kit to facilitate development of their own community-specific use cases and applications.

3 Infrastructure and Platform

In this chapter, we have a look at the DARE platform itself, focusing on the infrastructural components, and moreover, we specify the underlying cloud infrastructures at SCAI and GRNET including relevant hard- and software components.

To realize the DARE platform, several components needed to be brought together. Low-level infrastructural services needed to come together with the specific DARE services and tools as well as the domain-specific applications of the two use cases. In chapter 3.1, we hence give an overview over all platform components focusing on the infrastructural parts. In chapters 3.2- Testbed/Pre-Release Infrastructure and 3.3- Productive DARE Environment we describe the underlying resources provided for DARE, in the Clouds at GRNET (okeanos) and SCAI (Openstack). Finally, in 3.3 we take a closer look at the extent to which it was possible to collaborate with the initiatives of EOSC and make use of EOSC services.

3.1 Cloud-Ready Platform

The DARE stack, which builds on top of containerisation (Docker, Kubernetes) as well as parallelisation technologies (MPI), operates at the interface to the underlying compute resources as well as to the

developed user applications. Next to the independent "DARE components" realizing the DARE architecture, the platform also includes several supporting and general-purpose components providing generic functionality to the platform such as e.g. storage, monitoring and authentication. In the following overview, the individual components are briefly summarized.

The main DARE components are the following:

- **DARE Login Service:** interacts between DARE components and Keycloak. It exposes functionality for sign in to the platform including refreshing and validating a token.
- **dispel4py**: Python library for implementing workflows, especially stream-based workflows for data-intensive workflows but also generic sequences of executions of applications (Processing Elements in a Graph) with data flow between them (edges of the Graph). Maps execution on different execution environments, e.g. MPI, Spark or local execution with Python multiprocessing.
- **dispel4py-registry:** that acts as the registry of the available processing elements of dispel4py
- **CWL Workflow registry:** that acts as the registry for workflows based on CWL **s-ProvFlow:** that is responsible for collecting, preserving and reporting on provenance information from the workflow execution over the platform.
- **DARE Execution API:** enables the distributed and scalable execution of dispel4py and CWL workflows.
- **DARE Exec Registry:** was historically used to store and track experiments / executions on the DARE platform and its corresponding workspace. Has been merged into Exec API during the project.
- **DARE playground:** that builds an environment for testing and debugging, especially dispel4py workflows.
- **DARE Semantic Search:** allows to search through metadata in the Data catalogue using linked data principles
- **JupyterHub:** offering a pre-configured data science environment (customized for DARE)
- **Data Catalogue:** web service offering a simple interface to insert metadata about DARE data in the form of RDF into a triplestore based on OpenLink Virtuoso
- **Provenance Service (S-ProvFlow):** A collection of services including a Web API (provenance-api) and a front-end GUI (sprovflow-viewer) allowing acquisition, storage, exploration and visualization of provenance data which is produced at run-time by DARE Executions.
- **SemaGrow:** a federated SPARQL query processor that allows combining and cross-indexing remote datasources. It hides heterogeneity from client applications by federating SPARQL and non-SPARQL endpoints and meaning by mapping queries and query results between vocabularies.
- **Exareme:** A distributed and federated processing engine that provides a declarative language to define dataflows. Its algorithms are written in SQL with Python extensions to declaratively express data parallelism and complex computations using user-defined functions.

Apart from these, the platform also includes the following supporting and general-purpose components, that on the one hand build the infrastructural basis for the developed DARE components and on the other hand facilitate the execution of the main components:

- ***Docker/Kubernetes***: allowing for integration of containerized software components, container management and orchestration

- ***Rook ceph:*** providing the main distributed storage facility for the cluster, used to persist data between runs and store the users workspace
- ***Helm/Tiller:*** to help manage Kubernetes applications
- ***mpi-operator:*** that is used to enable the execution of MPI jobs in Containers as Kubernetes jobs
- ***cert-manager:*** enabling automatic x509 certificate management for Kubernetes services
- ***Ingress:*** acting as the main entry point to the platform, redirecting external requests to the corresponding pods running the applications such as DARE Execution or Provenance API.
- ***Keycloak:*** acting as the authentication and authorisation server, backend for the DARE login service
- ***Prometheus, Alertmanager*** **and** ***Grafana:*** for collecting and visualizing metrics of the cluster's performance for monitoring and operations.
- ***Harbor:*** ensures that the container images are free from vulnerabilities
- ***Mysql:*** acting as persistent storage of the d4p-registry
- ***Mongodb:*** acting as persistent storage of sprov
- ***Virtuoso:*** acting as persistent storage of the data-catalogue
- ***sprov-viewer:*** used as the frontend user interface for visualized provenance from sprov.
- **RabbitMQ:** used as a message queueing service for scalability of provenance insertion
- **SOLR/Lucene:** used for indexing of metadata terms for the DARE semantic search service

While the DARE components including the supporting systems rather belong to the scope of the other WPs and deliverables, WP5 was primary responsible for the layer building the infrastructural basis and framework for the DARE platform. This infrastructural framework consists of diverse tools that are illustrated in figure 1.

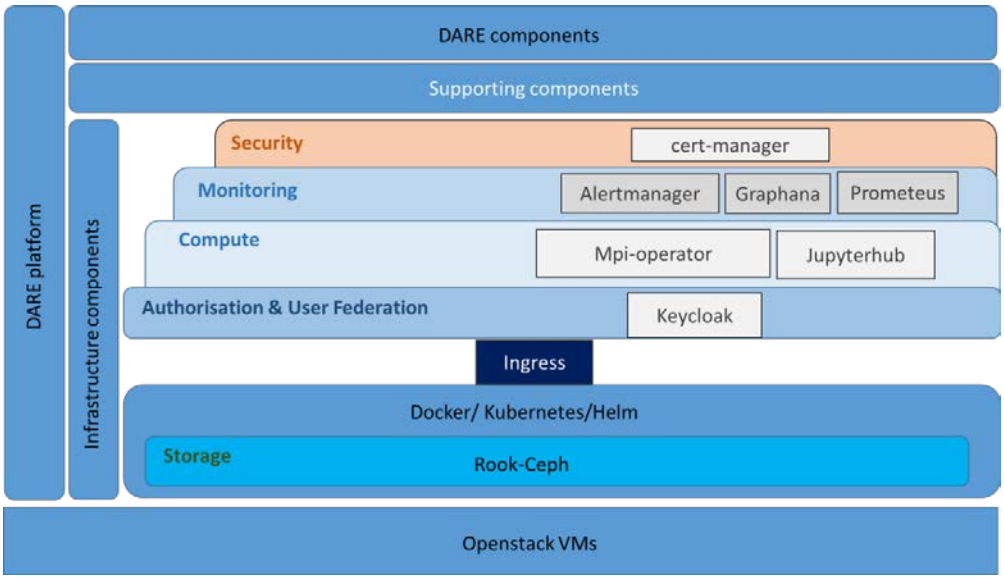

Figure 1: Cloud-ready DARE platform

Docker/Kubernetes-Universe

Already at the beginning, the project partners decided to rely on orchestrated micro services. Reason for this was among others, that a microservice-based architecture allows thinking about services as self-contained, independent applications, which facilitates their development. Also benefits like e.g. a more easy scaling and the possibility for more efficient system optimization and organization are advantages that make their use quite attractive.

For container management and orchestration, the project decided to use Kubernetes. The Kubernetes orchestration enables the effortless cloud deployment of the DARE platform and also the automated scaling and management of containerized applications (see chapter 4.1). Cluster management and deployment is operated through an API exposed by Kubernetes. This API enables external and internal communication exploiting user-client authentication. Kubernetes API additionally provides Role-Based Access Control (RBAC) that binds a user-client (which can be a containerized application) to the cluster. Using RBAC makes user authentication and access flexible and manageable by enabling permission control for Kubernetes resources. In DARE, the Kubernetes API is connected to the DARE Keycloak and roles are mapped from the Keycloak as well, so the DARE deployment provides a single point where permissions for Applications and the environment itself (Kubernetes) can be controlled.

One of the ideas of DARE is the enrollment of a complete infrastructure in the cloud including storage, which is not directly covered by Kubernetes itself. For this purpose, DARE makes use of Ceph[1]. Ceph is an open source, cloud-native and highly scalable distributed storage solution. To manage the Ceph deployment and integration into Kubernetes, DARE makes use of the Rook project[2]. Rook implements native Kubernetes operators which automate nearly all aspects of storage administration such as deployment of Ceph in Kubernetes, configuration, provisioning, upgrades and monitoring. It implements the Kubernetes Container Storage Interface (CSI) to make Ceph-based storage usable as normal Kubernetes volumes. Thus, Kubernetes applications are able to mount block devices and filesystems managed by Rook as native Kubernetes volumes. The CephFS filesystem additionally offers the functionality of a shared filesystem writable by multiple pods of the same time, in addition to the usual remote block device (RBD) based volumes that can be used by only one pod in read-write-mode at the same time. This CephFS filesystem is used in DARE to implement the workspaces that store the enduser files in DARE, e.g. to provide a shared filesystem for MPI computations where multiple pods need to access the shared directory between the running job pods. Figure 2 illustrates how Ceph Rook integrates with Kubernetes.

To facilitate the deployment and management of some components on the Kubernetes Cluster, Helm is used. Helm is a package manager for Kubernetes that allows developers and operators to more easily package, configure, and deploy applications and services onto Kubernetes clusters. There are two parts to Helm: the Helm client (helm) and the Helm server (Tiller).[3]

[1] https://ceph.io/
[2] https://rook.io/docs/rook/v1.2/
[3] https://v2.helm.sh/docs/install/

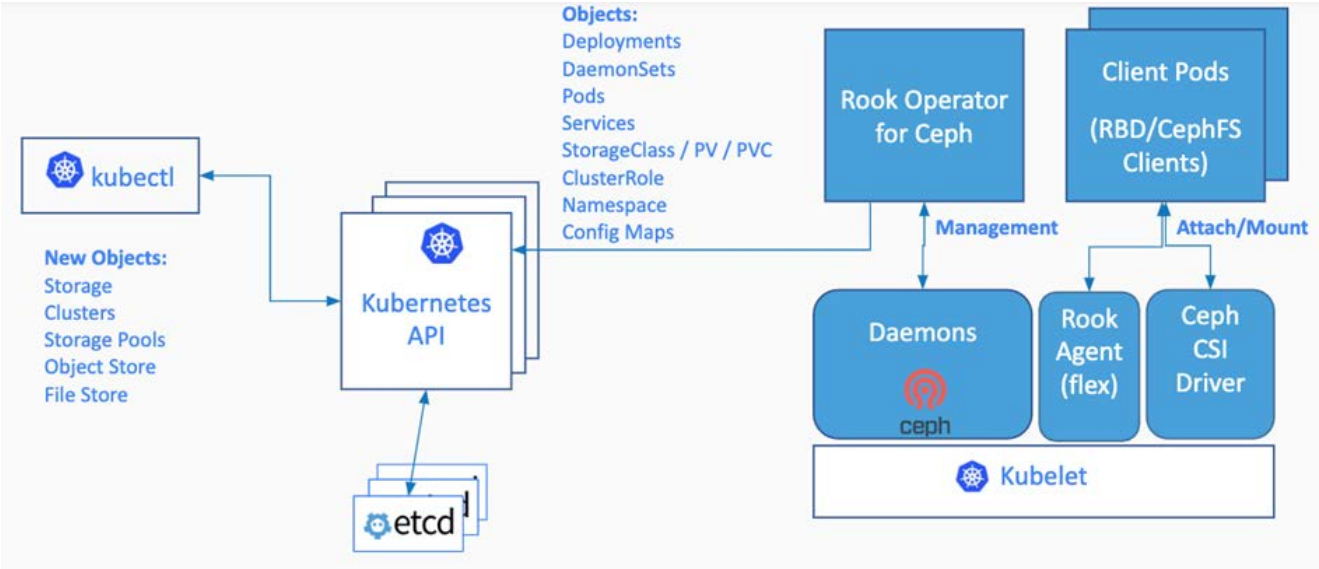

Figure 2: Kubernetes and Rook interactions[4]

Reverse Proxy:

To manage external access to the DARE platform, the Kubernetes Ingress[5] functionality is used, specifically the ingress-nginx controller[6]. Ingress exposes HTTP and HTTPS routes from outside to services within the DARE environment. Traffic routing is controlled by rules defined on the Ingress resource.

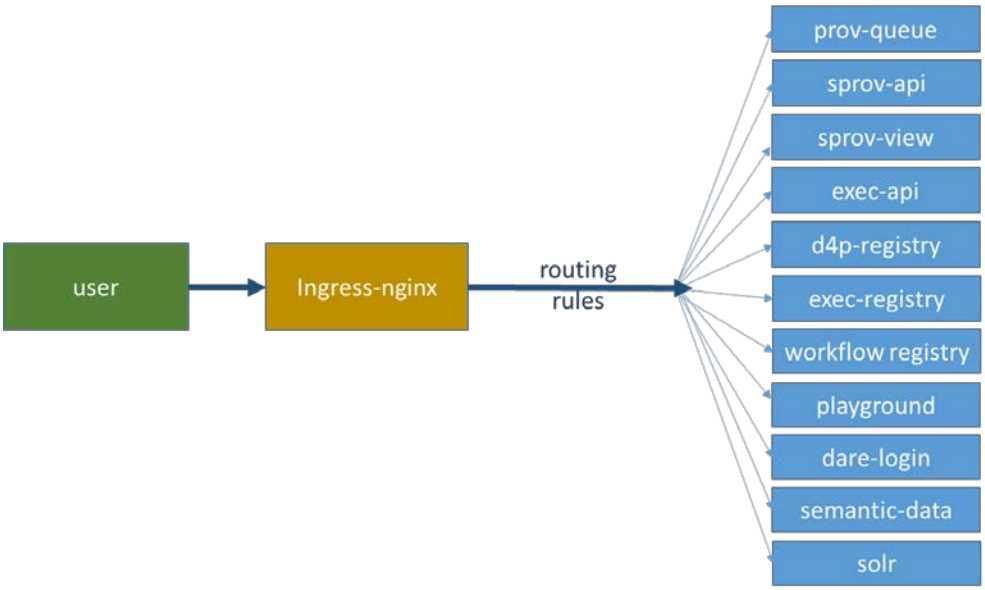

Figure 3: Ingress routing traffic to DARE services

Authentication and User Federation:

One major objective of WP5 was the provision of a reliable login service for the DARE platform. The authentication mechanism comprises the process of determining whether someone is, in fact, who he

[4] © Rook Authors 2020. Documentation distributed under CC-BY-4.0, **https://rook.io/docs/rook/v1.4/ceph-storage.html**
[5] https://kubernetes.io/docs/concepts/services-networking/ingress/
[6] https://github.com/kubernetes/ingress-nginx

declares himself to be. The authorization management, based on different roles, is needed to ensure that users with specific roles only have associated rights and only see data, relevant for their purposes.

To this end, we setup a single sign-on (SSO) environment based on Keycloak[7]. Keycloak is an "Open source identity and access management" solution, that allows to setup a central Identity Provider that applications (acting as Service Providers) use to authenticate and authorize user access. Furthermore, to seamlessly integrate with EOSC, the platform's authentication and authorization mechanisms have been designed to be interoperable with existing Authentication and Authorization Infrastructure (AAI). To that end, the DARE platform employs OAuth2 with OpenID Connect. The same technology is the basis for the EOSC portal[8], EGI Check-In[9] and EUDAT B2Access[10]. Through the use of an own Keycloak deployment, the DARE platform allows community administrators the choice to implement their own identity databases or integrate with all providers or proxies that implement the OAuth2 with OpenID Connect technology. These include the ones mentioned above as well as many other providers, such as Google, Facebook, GitHub, etc.

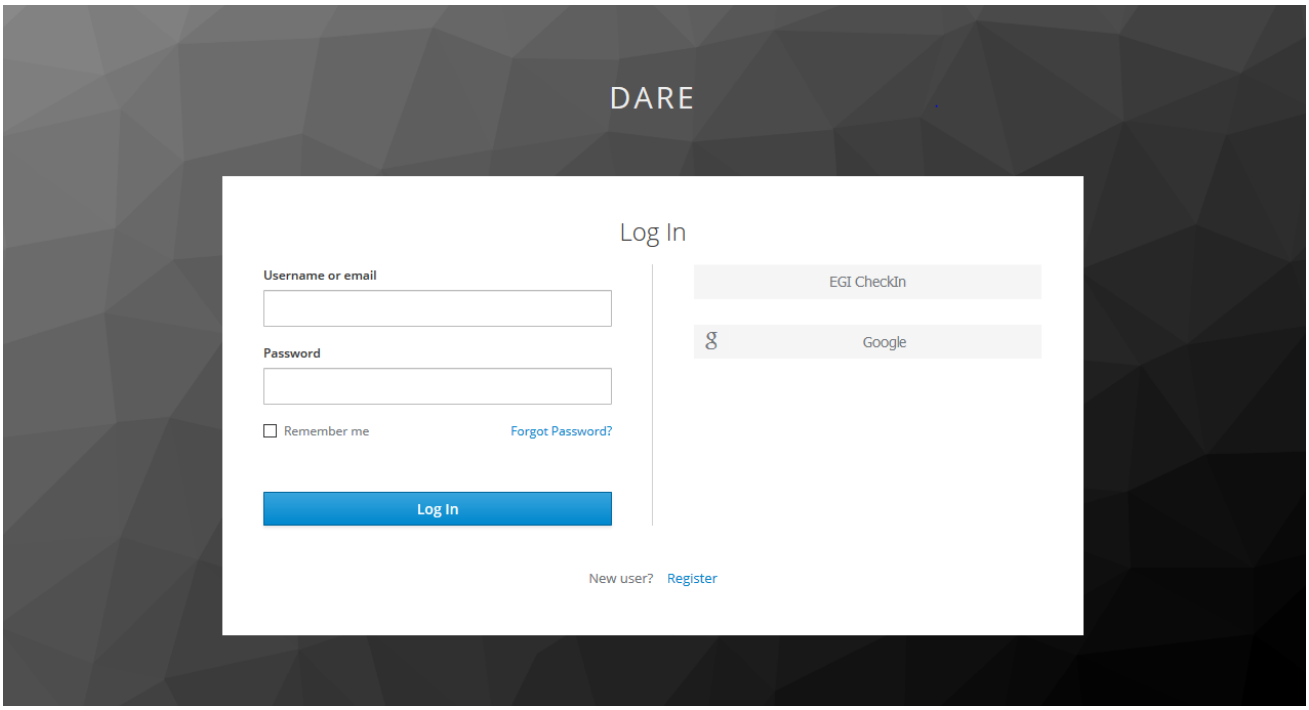

Figure 4: Keycloak Dashboard

This technology makes it also easy to couple with authenticating proxies (e.g. former Keycloak Gatekeeper, Lukeko or OAuth2 Proxy[11]) in Kubernetes sidecar containers, which are automatically injected with application containers on Kubernetes pods and act as a reverse proxy to the application. Connections to the service go to the sidecar proxy first, which checks if the user is already authenticated and if not sends a forward header to the login page. If the user is authenticated, the request is forwarded to the service with additional HTML headers injected, which allow the application to identify the user.

[7] https://www.keycloak.org/
[8] https://www.eosc-portal.eu/
[9] https://www.egi.eu/services/check-in/
[10] https://eudat.eu/services/b2access
[11] https://github.com/oauth2-proxy/oauth2-proxy

In the following, we present an exemplarily login flow:

1. A user wants to access a specific service via DARE´s JupyterHub
2. Clicking on the "Login" button causes that his request arrives at the reverse proxy. This service sits between the applications/services and the external user and forwards the requests to the appropriate service, in this case JupyterHub.
3. JupyterHub then sends the user to the Keycloak service. Keycloak is able to authenticate users e.g. with existing OpenID Connect or SAML 2.0 Identity Providers.
4. In the last step, Keycloak allows/not allows the access to the addressed service (JupyterHub)
5. The user token is loaded into users' notebook to provide SSO to the DARE services

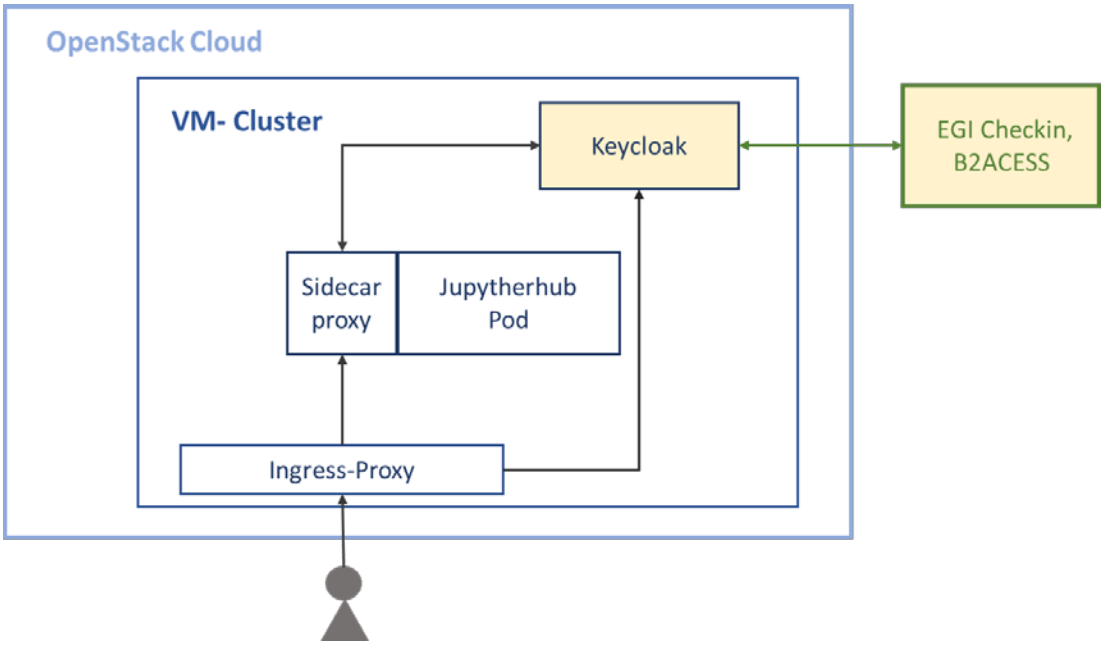

Figure 5: Example login flow

Compute:

In general, the DARE execution service API is a programmatic way to run and manage workflows based on the standard CWL[12] or dispel4py. To support parallel processing and enable the run of applications using MPI like e.g. SPECFEM3D[13] in the DARE cluster we use the MPI Operator, which is one of the core components of Kubeflow[14], a project aiming to make deployments of machine learning (ML) workflows on Kubernetes simple, portable and scalable.

Monitoring:

For monitoring and alerting, we use the open-source stack consisting of Prometeus[15], Alertmanager[16] and Grafana[17]. Prometheus, which scrapes and stores time series data, acts as the backend for Grafana, which is used as visualization software. The Alertmanager handles alerts sent by the Prometeus server and takes care of deduplicating, grouping, and routing them to the correct receiver. To implement this

[12] https://www.commonwl.org/
[13] https://geodynamics.org/cig/software/specfem3d/
[14] https://v0-2.kubeflow.org/docs/about/kubeflow/
[15] https://prometheus.io/
[16] https://prometheus.io/docs/alerting/latest/alertmanager/
[17] https://grafana.dare.scai.fraunhofer.de/

monitoring stack on the DARE cluster, individual components, manifests, Prometheus metrics and Grafana dashboards have been configured.

Security

For automated certificate management and issuance of TLS certificates we use cert-manager[18], which is native to Kubernetes. It is configured to automatically retrieve, manage, deploy and renew certificates for the external domains through the ACME protocol from the Let's Encrypt online Certificate Authority.[19] This relieves the administrators from the DARE deployment from nearly all burdens related to the management of certificates for secure TLS/HTTPS connections.

3.2 Testbed/Pre-Release Infrastructure

The cloud infrastructure for the testbed is deployed and maintained at GRNET's okeanos-knossos IaaS cloud. It currently consists of 6 VMs of two classes of resources.

The okeanos-knossos cluster[20] is a large IaaS cluster located at Crete, Greece and operated by GRNET. It is powered by Synnefo[21], an IaaS software with OpenStack-compatible APIs for compute, network and object storage.

A kubernetes cluster utilizes 3 VMs of 16 cores, 16GB RAM and 30 GB of permanent storage each and 40 GB extension each. The total storage space available for the kubernetes deployment is 210GB. A private network is also used to connect the 3 VMs. The kubernetes deployment consists of two masters (dare-kubernetes-1 and 2), while all three act as nodes.

The lifecycle of the testbed has been disrupted a few times during the development of the platform due to experimentations with immature or incomplete configurations and software. This was expected, though, as it was meant to be a punchbag for IT engineers, software developers and scientists contributing to the project, so it often had to be destroyed and rebuild over fresh resources.

There are also 3 VMs for auxiliary applications: a jumphost to access and manage the Kubernetes cluster, a testbed2-node1 to perform risky experiments related to the Kubernetes cluster and a VM to perform Authorization/Authentication experiments. Each of these is powered by 2 cores, 2MB of RAM and 30GB.

Access and utilization of the testbed resources was offered throughout the project development lifecycle. Collaborators were assisted with additional software whenever required (i.e. kamaki[22], kamaki-ansible-role[23]) to automate access to the IaaS cloud. Technical support and statistics where provided by the GRNET okeanos-knossos operators whenever requested.

[18] https://cert-manager.io/docs/
[19] https://letsencrypt.org/
[20] https://okeanos-knossos.grnetcloud.net
[21] https://synnefo.org
[22] https://www.synnefo.org/docs/kamaki/
[23] https://github.com/saxtouri/kamaki-ansible-role/

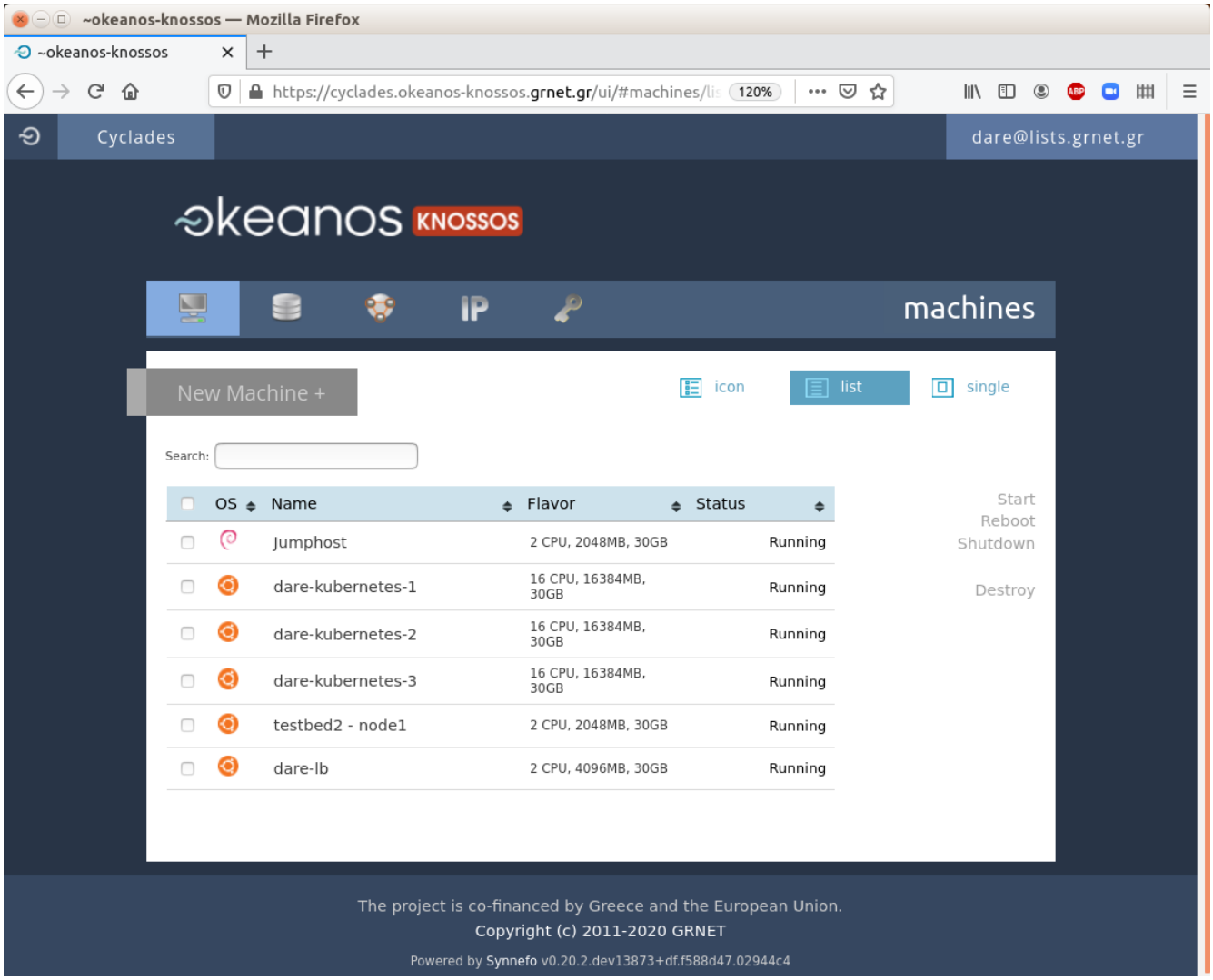

Figure 6: IaaS environment presenting the compute resources of testbed

3.3 Productive DARE Environment

The productive instance of DARE platform is orchestrated on SCAI´s OpenStack Cloud, which is part of the EGI Federated Cloud.

The permanent, for DARE reserved virtual resources provide a performance comparable with the performance of seven physical servers, equipped with 2x Intel Server CPUs@2.30GHz and 192GB RAM each. The Ceph storage system is hosted on the same nodes (hyper-converged) using Rook Ceph on 42 dedicated harddisks, offering a total of 241 TiB storage space for the DARE deployment. These resources were sufficient to serve up to 15 users running experiments and workflows in parallel during the performed training events and webinars. In case of trainings with more users, or higher compute intensive applications, or for major benchmarking tests, we were able to increase the provided DARE cloud resources.

Throughout the whole project period, SCAI provided technical support and consulting for the DARE partners and users, via direct contact and ticket system. Special support and consulting were provided for use case development and for the optimization of individual components.

During trainings and other events, the infrastructure and the access to it was monitored specifically and technical staff was available to be able to intervene in case of failure.

3.4 Relation to EOSC

Right from the beginning of the project, partners envisaged to fuse the DARE platform with the at that time newly arising landscape of the European Open Science Cloud. More specifically, this means that during the project duration partners continuously pursued all relevant upcoming technical solutions and services under the umbrella of EOSC and inspected to what extend these could be beneficial for the DARE project.
Services developed in the scope of European projects on the way to EOSC esp. EOSC-hub[24], which have proved to be useful, were already integrated in DARE. For example, the DARE login service is interoperable with EGI-Checkin[25] and for high volume data exchange B2Drop[26] is used.

In general, there is a strong connection between EOSC and the partners in DARE, which are e.g. also involved in EOSC-hub and other European projects, which may be considered as preparation projects for the EOSC. From that, a strong technical influence was presents and the DARE platform was as far as possible designed to be interoperable with upcoming and available EOSC services. Resources/VMS of SCAI and GRNET, used in DARE, are provisioned in the scope of the Federated Cloud (EOSC–HUB).

DARE intended to offer the final DARE platform on the EOSC marketplace[27], which offers easy access to operational services (provided by diverse providers) and resources for various research domains. While it was considered a great opportunity to present the DARE platform to other scientific communities, it was found in discussion with related EOSC projects that the marketplace currently does not offer the correct general framework to publish the kind of result of DARE (i.e., a kind of SDK instead of a running service).

4 Platform Deployment

For the development and optimization of the platform that is adapted to the requirements of the two use cases, DARE used a deployment strategy that was already introduced in D5.1. In the following chapter 4.1, we will reiterate this deployment strategy and adapt it to the latest status. In chapter 4.2, we present a detailed schedule of all platform releases within the course of the project. Finally, in chapter 4.3 we shortly describe how the automated deployment of the final and packaged platform release was realized. Related deployment guidelines for users are provided in D5.4.

4.1 Deployment Strategy

In close collaboration with the other work packages, a process was defined to standardise the development of all DARE components and guide the testing cycles. The deployment process for new releases of the DARE platform can be summarized in six steps, visualized in figure 1.

[24] https://www.eosc-hub.eu/
[25] https://www.egi.eu/services/check-in/
[26] https://eudat.eu/services/b2drop
[27] https://marketplace.eosc-portal.eu/

Figure 7: Deployment pipeline

(1) Software Development and first Testing

For the development and first testing (e.g., component testing, integration testing) of the individual software products and platform components, WP5 set up a software development platform/pre-release environment, which was continuously adjusted to meet the requirements of the development teams. This environment is provided in GRNETs cloud infrastructure okeanos[28] and was available for DARE developers from the beginning of the project.

(2) Provision of Docker[29] and Kubernetes[30] Files

As described in chapter 3.1, the project decided to rely on orchestrated micro services. For container management and orchestration, the project decided to use Kubernetes, which is the most prominent among the orchestration tools and widely used in both research and industry. It enables automated deployment, scaling and management of containerized applications and exposes information on the status and availability of each container via its native API, which facilitates operation and management of the deployment.

Therefore, after the successful development of the individual services, (see 1. in Fig.1) the next step was the provisioning of related Docker and Kubernetes files (see 2. In Fig.1). To give a short insight, in the following we exemplarily present the Docker and Kubernetes files for the DARE login service[31].

A dockerfile is a text document containing all commands a user would have to call on the command line to assemble a specific image. By using the dockerfile an automated build is created, that successively executes all pre-configured instructions and thereby creates the required image to initiate a container later.

Figure 2 shows a cut-out of the dockerfile for the configuration of the DARE login-service, based on the nginx-proxy image and published in the respective gitlab repository at https://gitlab.com/project-dare/dare-platform/-/blob/master/containers/dare-login.

[28] https://okeanos.grnet.gr/home/
[29] https://www.docker.com/get-started
[30] https://kubernetes.io/
[31] https://project-dare.gitlab.io/dare-platform/api/

Dockerfile 2.17 KB

```
FROM nginx

RUN apt-get update && apt-get install -y build-essential supervisor unzip uwsgi python3 python3-pip python3-s

ENV HOME=/opt/repo

# Copy the application
ARG RELEASE=v1.1

RUN git clone --branch $RELEASE --single-branch https://gitlab.com/project-dare/dare-login.git
RUN python3 -m pip install -r ./dare-login/requirements.txt
RUN mkdir ${HOME}/src
RUN cp ./dare-login/app/* ${HOME}/src

RUN rm -rf dare-login/

# Copy the configuration file from the current directory and paste
# it inside the container to use it as Nginx's default config.
COPY ./nginx.conf /etc/nginx/nginx.conf

# setup NGINX config
RUN mkdir -p /spool/nginx /run/pid && \
    chmod -R 777 /var/log/nginx /var/cache/nginx /etc/nginx /var/run /run /run/pid /spool/nginx && \
    chgrp -R 0 /var/log/nginx /var/cache/nginx /etc/nginx /var/run /run /run/pid /spool/nginx && \
    chmod -R g+rwX /var/log/nginx /var/cache/nginx /etc/nginx /var/run /run /run/pid /spool/nginx && \
    rm /etc/nginx/conf.d/default.conf

# Copy the base uWSGI ini file to enable default dynamic uwsgi process number
COPY ./uwsgi.ini /etc/uwsgi/apps-available/uwsgi.ini
RUN ln -s /etc/uwsgi/apps-available/uwsgi.ini /etc/uwsgi/apps-enabled/uwsgi.ini

COPY ./supervisord.conf /etc/supervisor/conf.d/supervisord.conf
RUN touch /var/log/supervisor/supervisord.log

EXPOSE 80:80
```

Figure 8: dockerfile for DARE login-service[32]

After the build of the dockerfiles, the service images were made available for usage by pushing them to the Gitlab Container Registry of the DARE development project.

Next, the Kubernetes files, defining and organizing the container orchestration across the cluster were created. Depending on the service, different files were generated. For the DARE-login service e.g. a deployment[33] and a service[34] descriptor have been created.

A deployment descriptor describes the desired state of a deployment, i.e. how and where certain containers should be running on a Kubernetes cluster. There are several parameters described, such as which container images to run where, the number of pods to be started, and the way in which they should be updated. Altogether, the deployment file describes the whole life cycle of the deployment. A cut-out of the deployment descriptor of the DARE-login service is presented in figure 9.

[32] https://gitlab.com/project-dare/dare-platform/-/blob/master/containers/dare-login/Dockerfile
[33] https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
[34] https://kubernetes.io/docs/concepts/services-networking/service/

dare-login-dp.yaml 1.26 KB Edit Web IDE

```yaml
kind: Deployment
apiVersion: apps/v1
metadata:
  name: dare-login
  labels:
    app: dare-login
spec:
  replicas: 1
  selector:
    matchLabels:
      app: dare-login
  template:
    metadata:
      labels:
        io.kompose.service: dare-login
        app: dare-login
    spec:
      containers:
      - name: dare-login
        image: registry.gitlab.com/project-dare/dare-platform/dare-login:v1.1
        imagePullPolicy: Always
        ports:
        - containerPort: 80
```

Figure 9: deployment descriptor of DARE-login service[35]

A service in Kubernetes, defined through a service descriptor, is an abstraction, which allows you to expose an application running on a set of pods to the network, e.g. it exposes a web service interface under a certain name and port to other pods running in the Kubernetes cluster or the outside world (or both). Exemplarily in figure 10 the service descriptor of the DARE-login service is shown.

dare-login-svc.yaml 200 Bytes Edit Web IDE

```yaml
kind: Service
apiVersion: v1
metadata:
  name: dare-login
  labels:
    name: dare-login
spec:
  ports:
  - name: http
    port: 80
    targetPort: 80
    protocol: TCP
  selector:
    app: dare-login
```

Figure 10: service descriptor of DARE-login service[36]

For other services, the provision of further files is needed. E.g. the d4p-registry, which includes a database, needs a further file for claiming a persistent volume (see figure 11).

d4p-registry-db-ss.yaml 995 Bytes Edit Web IDE

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
   name: d4p-registry-db-pvc
spec:
   accessModes:
   - ReadWriteOnce
   resources:
     requests:
       storage: 5Gi
---
```

[35] https://gitlab.com/project-dare/dare-platform/-/blob/master/k8s/dare-login-dp.yaml
[36] https://gitlab.com/project-dare/dare-platform/-/blob/master/k8s/dare-login-svc.yaml

Figure 11: PersistentVolumeClaim for d4py-registry[37]

(3) Deployment to Testbed

After the creation of the above-mentioned files, the newly developed software releases were deployed on the Pre-Release Testbed environment at GRNET, which was designed similar to the productive platform environment provided on Cloud resources at SCAI.

Using the Kubernetes API, it was possible to manually create the deployment e.g. for the DARE login service by running the following commands:

1. To deploy the deployment file.

```
kubectl apply -f https://gitlab.com/project-dare/dare-platform/-/blob/master/k8s/dare-login-dp.yaml
```

2. To deploy the service file.

```
kubectl apply –f https://gitlab.com/project-dare/dare-platform/-/blob/master/k8s/dare-login-svc.yaml
```

To check whether, the deployment was correctly created and all pods are up and running, Kubernetes provides simple commands[38] like:

Table 1: Exemplary Kubernetes commands

`kubectl get deployments`	displays the status of deployments
`kubectl get pod`	displays the name and status of pods in the default namespace of the Kubernetes environment
`kubectl get pod –o wide`	additionally provides information about the orchestration of the pods over the Kubernetes nodes

Next to the manual deployment, partners were also provided with the possibility to use the gitlab-ci[39], a tool build into the Gitlab, which continuously (once a day) performed automated deployments and pre-defined unit tests in the Kubernetes environment of the Pre-Release Testbed environment. In case of error messages or if certain problems become apparent during the deployment process, issues were reported via GitLab. As an example figure 12 shows the `gitlab-ci.yml` file, configuring the checking pipeline for the s-ProvFlow service.

[37] https://gitlab.com/project-dare/dare-platform/-/blob/master/k8s/d4p-registry-db-ss.yaml
[38] https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands
[39] https://docs.gitlab.com/ee/ci/

.gitlab-ci.yml 2.59 KB Edit Web IDE

```yaml
image: docker:latest

variables:
  DOCKER_DRIVER: overlay2

services:
  - docker:dind

stages:
  - test
  - build

unit_tests:
  stage: test
  script:
    - apk add --update --no-cache python3 py3-pip py-cryptography gcc g++ libxml2-dev libxslt-dev python3-dev # Requirement for lxml
    - cd provenance-api
    - pip3 install --upgrade unittest-xml-reporting coverage
    - pip3 install --upgrade -r requirements.txt
    - cd src/test/
    - mkdir coverage_report
    - coverage run --source=../. --omit=test* --rcfile=.coveragerc --branch -m unittest discover
    - coverage report
    - coverage html --directory=coverage_report/
    - cp -r $(pwd)/coverage_report/ $CI_PROJECT_DIR/coverage_report
  coverage: '/^COVERAGE:(.*%)$/'
  artifacts:
    expose_as: 'coverage report'
    paths: ['coverage_report/']
  only:
    - branches
  except:
    - master
```

Figure 12: gitlab-ci.yml for s-ProvFlow[40]

(4) Application Acceptance Tests

Following these simple build and unit tests, further functionality and security tests were performed before the release was finally provided in the productive DARE platform. Functionality tests were performed to ensure that the applications of the involved use-cases work and meet the expectations of the domain specific communities. Therefore, in the first step the software developers verified that all software components and features as well as complete workflows act in accordance with pre-determined technical and functional requirements.

For the basic platform components, which include the dare login service, the two registries, the Execution API and the sprov API, the DARE partners e.g. developed specific tests, which are provided in DAREs Gitlab repository. The tests were used to check the deployed services, running in the productive DARE environment, but could also be used by future users to test local DARE installations. Instructions for the individual tests are provided in the corresponding ReadMe.md files (https://gitlab.com/project-dare/dare-platform/-/tree/master/test_dare_components).

Additionally, partners also performed benchmarks e.g., of s-ProvFlow API to compare the difference in overhead imposed on services using the s-ProvFlow API with the provenance insert queue enabled or disabled. This helped to understand the general behavior of s-ProvFlow and encouraged tuning possibilities (see https://gitlab.com/project-dare/s-ProvFlow/-/tree/master/docker/benchmark).

Next to these manual tests, in a second step, the DARE services and packages were tested in the scope of the use case WPs, where the developers implemented their workflows based on them. In this way it was possible to test whether the packages and services are complete, functional and provide correct results.

[40] https://gitlab.com/project-dare/s-ProvFlow/-/blob/master/.gitlab-ci.yml

In a third step, real users were consulted, to test the DARE platform and to measure how intuitive and helpful the developed tools are. To this end, DARE partners organized training events with users from the climate and seismology community, which allowed the partners to gather feedback for the next development phase. More detailed information and the analysis of feedback is summarized in the deliverables of the related work packages WP6, WP7 and WP8, in D6.3, D6.4, D7.3, D7.4, D8.4 and D8.5.

In the last step, simple security testing was performed. For security testing of the Kubernetes and Docker deployments, WP5 set up a Harbor[41] installation. As soon as ready Docker images were marked for security check, the images were uploaded to the Harbor registry reachable at https://harbor.dare.scai.fraunhofer.de. The Harbor server scanned the images for possible vulnerabilities, and finally if no security issues occurred, signed them as trusted. Trusted images were finally deployed in the production environment of DARE platform (see (5)).

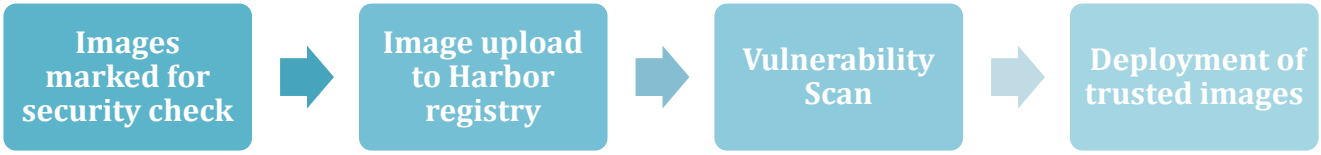

Figure 13: DAREs vulnerability scans with Harbor

(5) Deployment to Productive Environment

After the successful conclusion of this testing phase, in the fifth step the components were published to the production platform. More specifically, this means that the final images provided through the corresponding gitlab repository were pushed to the Kubernetes environment provided at SCAI, which is described in chapter 3.3. The deployment procedure runs analogous to the deployment in (3).

(6) Post Deployment Tests

Next to the application testing in (4), partners also performed stress and load testing to assess performance, stability and effectiveness of the basic components. For example, we checked individual components with tools like “Apache benchmark”, to verify that all services have acceptable average request times.

Moreover, in advance of training events, test-user accounts simultaneously running jobs were used to test whether the system is stable and jobs finish successfully. In training events with hands-on sessions like the volcanology summer school at KIT on 24 July 2020, the Climate and Atmospheric Sciences webinar on 16 October 2020 and the seismological webinar on 11 November 2020, real users tested the operational DARE environment not only for usability but also for stability and robustness. Further information on the infrastructure testing is described in chapter 5 - Usage of the DARE Platform environments.

4.2 Deployment Schedule

As already described in D5.1, we distinguished between an internal and a public deployment. While the internal deployment (only including steps (1)-(4) of the deployment strategy) only happened on the

[41] https://goharbor.io/

DARE Pre-release testbed at GRNET to ensure a rapid progress, the public deployment included the provision of new platform releases on the production platform environment at SCAI.

In the following table, we summarized all public platform releases. The releases include the four major platform releases v1.0, v2.0, v3.0 and v3.6 as well as all minor releases that took place in between.

As described in the deployment strategy, each platform release that was published in the corresponding GitLab Repository was immediately followed by a platform deployment to the productive DARE environment. Hence, each major platform release represents one of the four milestones MS3 (1st Platform Deployment), MS6 (2nd Platform Deployment), MS10 (3rd Platform Deployment), MS12 (final Platform Deployment).

Table 2: DARE platform releases

October 2019	v1.0	DARE Execution API was integrated in the DARE platform. All necessary files for Docker and Kubernetes were added to the DARE platform release.
January 2020	v2.0	A new testing environment was included in the platform. Additionally, the DARE platform's shared file system was re-organized to be more user oriented. Users can view/download their generated files by interacting with the DARE's execution API.
February 2020	v2.1	In this version, a queue service was added to unburden the workflow executions/insert sprov-api end point. Additionally, CWL provenance to s-prov mapping has been improved and sprovflow-viewer search options have been expanded. Now it is possible to add more search terms in the filter dialog.
April 2020	v3.0	This version offers innovations in almost all major platform components. First, the new testing environment (playground module) was updated to run in a nginx server (instead of gunicorn) and was completely integrated with the AAI. In a similar way, the Execution API is now based on nginx image and was integrated with the AAI. Also, the Provenance component had some important changes, i.e., a more resilient queue, unit tests on ProvenanceStore class, new search expression on API and in GUI. Finally, the dispel4py library includes the following changes: • The -f and -d arguments can be used to define the workflow inputs as shown in the workflow provenance. The -f argument accepts a path to a file containing the input dataset in JSON format. The -d argument accepts the input dataset in JSON format. • The -f argument has priority over the -d argument. • Fixed the issue inline provenance definition in the workflow script is used to create the workflow provenance when the argument –provenance-config is present. The –provenance-config has priority over the inline provenance definition now. • Attention: "s-prov:WFExecutionsInputs" in –provenance-config is deprecated.

		Furthermore, a completely new component, namely the Execution Registry, has been integrated in the platform. This component serves as backend to the Execution API handling the Shared File System, and more specifically, the experiments & runs of the users as well as their uploads.
May 2020	v3.1	From this version on, apart from dispel4py, CWL workflows are supported. Moreover, a new component was deployed in the platform, i.e., the workflow-registry. Now, domain developers can register their Docker containers and CWL workflows in the CWL Workflow Registry and refer to them by name and version. Client-side functions are provided, which wrap all the endpoints provided by the workflow-registry component. These functions can be used to store, update, retrieve and delete Dockers and workflows. Users can download and modify a demo jupyter notebook (here) in order to register their own Dockers and workflows. For the CWL execution, research developers can use the /run-cwl endpoint of the Execution API, specifying the name and version of the workflow. Thus, the DARE platform will deploy the execution environment and run the workflow. As usual, logs and output files are stored in the platform's Shared File System.
May 2020	v3.2	In the v3.2 release, the Keycloak AAI has been completely integrated to the Dispel4py Information Registry. Moreover, various updates in the Execution API were performed. We have also deployed a new component, as a central login point, which handles all the sign in actions and validates the provided access tokens. The dare-login component uses Keycloak as backend to issue tokens and validate the users. The rest DARE components use the dare-login for all AAI actions. As usual, client-side helper functions to interact with the component as well as a GitLab page for technical documentation were provided.
June 2020	v3.3	In this release multiple issues were fixed, especially regarding the CWL support.
July 2020	v3.4	The v3.4 release contains various fixes and updates in workflow execution and monitoring. The dare-login and exec-api components were improved to handle more requests at the same time during the monitoring of the workflow execution.
October 2020	v3.5	The v3.5 release contains various fixes and updates in the provenance component for the CWL workflows. It also contains fixes in the Execution API and its helper functions as well as integration tests for the components' APIs.
November 2020	v.3.6	The v3.6 release is the final release and contains various fixes and updates in the Execution API component. It's written almost from scratch, has improved features and easier API. The two Shared File Systems are now integrated into a single one and all the DARE use cases are updated and follow the same folder structure (see e.g. Deliverable D6.4).

4.3 Automation of Deployment

The DARE stack has been designed and packaged so that it is easily deployable on a cloud e.g. in EOSC infrastructure services. In detail this means that DARE offers ready-to-use infrastructure descriptions and deployment recipes based on the well-known Ansible[42] by Red Hat and Terraform[43] by HashiCorp. Terraform is used to automatically deploy a set of Virtual Machines with the required infrastructure properties such as CPU, memory, storage and network interfaces on an IaaS Cloud of choice. Supported Cloud back ends include e.g., AWS, Microsoft Azure and OpenStack.
After the Virtual Machines have been started, Ansible installs and configures the Kubernetes stack, and starts complementing add-ons, such as networking (based on Calico[44]) and storage (based on Rook Ceph[45]).
Another possiblility to install the needed Kubernetes cluster on e.g. an Openstack cloud or on EOSC is to make use of Kubespray[46], which is a composition of Ansible playbooks, inventory, provisioning tools, and domain knowledge for generic OS/Kubernetes clusters configuration management tasks.[47]
The DARE stack running on top of Kubernetes in turn makes extensive use of Helm Charts to package and manage the Kubernetes resources and applications. This facilitates their usage on Kubernetes clusters.
The related deployment guidelines including best practices for managing and maintaining a DARE deployment on the cloud are provided in D5.4- Operational Requirements and Guidelines II.

5 Usage of the DARE Platform environments

During the course of the project, the DARE platform environments primary served for three purposes. On the one hand as development platform for software and infrastructural components, on the other hand as testing and training platform to verify functionality and to gather feedback from users of the two use case communities. Moreover, it was used to promote the project and its results. While the trainings performed in 2019 primary focused on feedback referring the usability of workflows and features, the trainings in 2020, that included hands-on sessions, additionally aimed to test functionality and performance of the underlying infrastructure.

To the public the operational platform environment served as a demonstrator and was not open for external users. After the project, the platform in the form of an easy deployable software package is widely available for researchers and their applications.

Development:
The major purpose of the DARE platform environments, especially in the first phase of the project was to create a space for the development of the individual DARE components and the integration of the DARE stack. Regarding the infrastructure architecture and the choice of individual infrastructural components, this e.g. included an evaluation and testing of container orchestration tools like Docker Swarm and Kubernetes. Due to its rather simple handling, at the beginning of the project, Docker swarm was the first choice for the DARE platform. However, during the project duration, Kubernetes

[42] https://www.ansible.com/
[43] https://www.terraform.io/
[44] https://www.projectcalico.org/calico-networking-for-kubernetes/
[45] https://rook.io/docs/rook/v1.4/ceph-storage.html
[46] https://kubernetes.io/docs/setup/production-environment/tools/kubespray/
[47] https://kubernetes.io/docs/setup/production-environment/tools/kubespray/

became more attractive and evolved to the industry standard for orchestration. Therefore, DARE partners performed an extensive evaluation of Kubernetes and finally decided to switch, especially because of its simple integration with ceph storage (see Rook) and Keycloak.

Also, for the decision on the best suitable solutions for AA, partners evaluated several systems and protocols like Unity[48], Perun[49] and Keycloak. Due to the fact, that Keycloak is widely used in most European infrastructure projects, in research and in industry and because of many useful features (support of OpenID/Oauth2) it became the preferred candidate. A more detailed discussion on this is provided in D2.2, chapter 5.2.

To take into account the performance requirements of HPC-applications in a Kubernetes environment we tried to run message passing (MPI) applications like SPECFEM3D on different container network options. In this context, network performance is an important aspect, and the project evaluated several network solutions. First, partners tested Flannel[50], which however seemed instable in our implementations. Following, Weave Net[51] and Calico were evaluated. Finally, by weighting up between performance and functionality, DARE decided to stick to the popular Calico-plugin[52] from Calico Project[53].

Testing & Trainings:
To verify functionality, assess performance and stability of the platform, the partners performed simple tests with tools like e.g. Apache benchmark[54] to check that the infrastructure components like e.g. Jupyterhub and Keycloak behave properly and have acceptable average request times. Exemplarily, in figure 14 the CPU usage during the testing of Keycloak and Jupytherhub is showcased, in a scenario with 500.000 requests and a concurrency of 150.

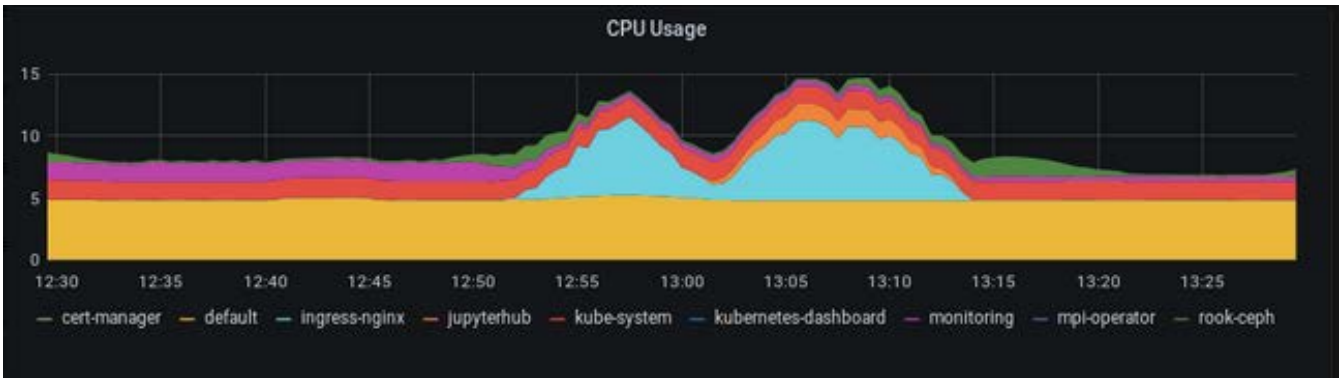

Figure 14: CPU usage during Apache benchmark test

A similar test is presented in figure 15, which highlights the CPU usage during a “long-term” stress test with 5mio requests to both portals and a concurrency of 600.

[48] https://unity.com
[49] https://perun.network/
[50] https://kubernetes.io/docs/concepts/cluster-administration/networking/#flannel
[51] https://www.weave.works/docs/net/latest/overview/
[52] https://github.com/projectcalico/cni-plugin
[53] https://www.projectcalico.org/calico-networking-for-kubernetes/
[54] https://httpd.apache.org/docs/2.4/programs/ab.html

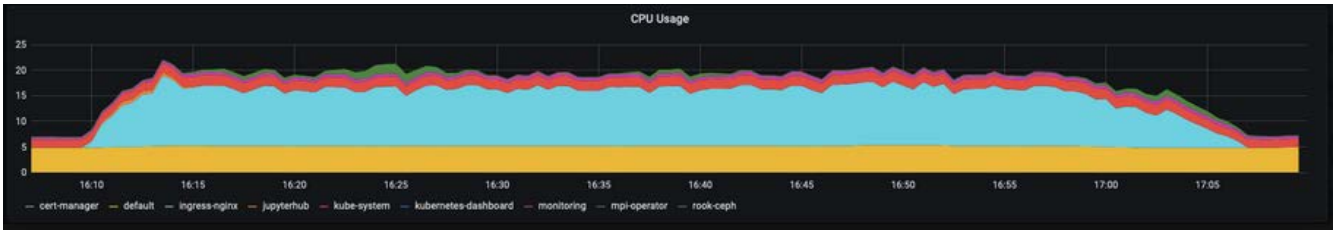

Figure 15: CPU usage during "long term" stress test

Both tests have shown that, in both scenarios, the CPU resources are not fully utilised.

In addition to such standard tests and in advance of training events partners also performed more realistic tests and used test-user accounts simultaneously running jobs to test whether the system behaves stabile and the jobs finish successfully. E.g., prior to the volcanology summer school at KIT on 24 July 2020 partners performed a test with the following steps:

1. Twelve test-users execute in parallel the simulation step (each user has a launcher and a worker container)

```
NAME                              READY   STATUS    RESTARTS   AGE
cwl-0a041b9462-launcher-z66rx     1/1     Running   0          2m20s
cwl-0a041b9462-worker-0           1/1     Running   0          2m28s
cwl-0fb8d3c5df-launcher-tlmz2     1/1     Running   0          119s
cwl-0fb8d3c5df-worker-0           1/1     Running   0          2m2s
cwl-3268151e52-launcher-jjpqn     1/1     Running   0          2m6s
cwl-3268151e52-worker-0           1/1     Running   0          2m9s
cwl-5269ef980d-launcher-7pcbs     1/1     Running   0          2m15s
cwl-5269ef980d-worker-0           1/1     Running   0          2m19s
cwl-5860faf02b-launcher-w9tfh     1/1     Running   0          2m17s
cwl-5860faf02b-worker-0           1/1     Running   0          2m22s
cwl-5a39bead31-launcher-m2q7r     1/1     Running   0          2m11s
cwl-5a39bead31-worker-0           1/1     Running   0          2m16s
cwl-5bbf1a9e0d-launcher-clrhh     1/1     Running   0          114s
cwl-5bbf1a9e0d-worker-0           1/1     Running   0          117s
cwl-6025d18fe4-launcher-n67q8     1/1     Running   0          2m18s
cwl-6025d18fe4-worker-0           1/1     Running   0          2m25s
cwl-81115e31e2-launcher-ms8rs     1/1     Running   0          109s
cwl-81115e31e2-worker-0           1/1     Running   0          113s
cwl-bd35283fe8-launcher-6bzvx     1/1     Running   0          105s
cwl-bd35283fe8-worker-0           1/1     Running   0          109s
cwl-ecb48a1cc9-launcher-k72s9     1/1     Running   0          2m9s
cwl-ecb48a1cc9-worker-0           1/1     Running   0          2m13s
cwl-f60afa4989-launcher-wg9zj     1/1     Running   0          2m3s
cwl-f60afa4989-worker-0           1/1     Running   0          2m5s
```

3. Resulting netcdf files for all users have successfully been produced.

```
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user1/uploads/
simres_grids.nc  test.json
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user2/uploads/
simres_grids.nc  test.json
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user3/uploads/
simres_grids.nc  test.json
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user4/uploads/
simres_grids.nc  test.json
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user5/uploads/
simres_grids.nc  test.json
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user6/uploads/
simres_grids.nc  test.json
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user7/uploads/
simres_grids.nc  test.json
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user8/uploads/
simres_grids.nc  test.json
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user9/uploads/
simres_grids.nc  test.json
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user10/uploads/
simres_grids.nc  test.json
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user11/uploads/
simres_grids.nc  test.json
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user12/uploads/
simres_grids.nc  test.json
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs#
```

3. For every user the execution log files exist.

```
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user1/runs/user1_20200717151111_7418e05951c443b4b954b9039af6541a/output
logs.txt
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user2/runs/user2_20200717151114_135ea4858e59472ea6efa840868f860e/output
logs.txt
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user3/runs/user3_20200717151117_bf3f46e0a6704ea0ae4ac66b7f89121f/output
logs.txt
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user4/runs/user4_20200717151119_04b7cda17d4546d8bdf37448a82fa374/output
logs.txt
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user5/runs/user5_20200717151123_78935baf28ba438a8e8b6ca0d29095ff/output
logs.txt
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user6/runs/user6_20200717151126_46e2163f48a54a35aae7f343ed68e48f/output
logs.txt
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user7/runs/user7_20200717151129_06d6e1fbd5d140ff8b8afe9ee6c41110/output
logs.txt
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user8/runs/user8_20200717151133_c9b88e5dac8b425e8d9293062d75a1db/output
logs.txt
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user9/runs/user9_20200717151136_3c0793260f4846cd8ad77b7dafd84072/output
logs.txt
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user10/runs/user10_20200717151140_e9d8fe6bd1874fa580446e9007d5cb20/output
logs.txt
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user11/runs/user11_20200717151145_68d102dce5cf48e697930e08e6fd19b1/output
logs.txt
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user12/runs/user12_20200717151149_5fcf82e31da9414eaa3b6cac73642c8e/output
logs.txt
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs#
```

4. All users simultaneously perform the plotting step (each user has a launcher and two worker containers)

```
NAME                                    READY   STATUS    RESTARTS   AGE
d4p-openmpi-0a041b9462-launcher-jt5h5   1/1     Running   0          4m44s
d4p-openmpi-0a041b9462-worker-0         1/1     Running   0          4m48s
d4p-openmpi-0a041b9462-worker-1         1/1     Running   0          4m48s
d4p-openmpi-0fb8d3c5df-launcher-gtgzk   1/1     Running   0          4m10s
d4p-openmpi-0fb8d3c5df-worker-0         1/1     Running   0          4m23s
d4p-openmpi-0fb8d3c5df-worker-1         1/1     Running   0          4m23s
d4p-openmpi-3268151e52-launcher-5kv6c   1/1     Running   0          4m11s
d4p-openmpi-3268151e52-worker-0         1/1     Running   0          4m30s
d4p-openmpi-3268151e52-worker-1         1/1     Running   0          4m30s
d4p-openmpi-5269ef980d-launcher-ms9k9   1/1     Running   0          4m37s
d4p-openmpi-5269ef980d-worker-0         1/1     Running   0          4m40s
d4p-openmpi-5269ef980d-worker-1         1/1     Running   0          4m40s
d4p-openmpi-5860faf02b-launcher-jbnfx   1/1     Running   0          4m38s
d4p-openmpi-5860faf02b-worker-0         1/1     Running   0          4m43s
d4p-openmpi-5860faf02b-worker-1         1/1     Running   0          4m43s
d4p-openmpi-5a39bead31-launcher-frvnh   1/1     Running   0          4m33s
d4p-openmpi-5a39bead31-worker-0         1/1     Running   0          4m37s
d4p-openmpi-5a39bead31-worker-1         1/1     Running   0          4m37s
d4p-openmpi-5bbf1a9e0d-launcher-58qvn   1/1     Running   0          4m9s
d4p-openmpi-5bbf1a9e0d-worker-0         1/1     Running   0          4m19s
d4p-openmpi-5bbf1a9e0d-worker-1         1/1     Running   0          4m19s
d4p-openmpi-6025d18fe4-launcher-gnvnb   1/1     Running   0          4m42s
d4p-openmpi-6025d18fe4-worker-0         1/1     Running   0          4m45s
d4p-openmpi-6025d18fe4-worker-1         1/1     Running   0          4m45s
d4p-openmpi-81115e31e2-launcher-f86n2   1/1     Running   0          4m9s
d4p-openmpi-81115e31e2-worker-0         1/1     Running   0          4m14s
d4p-openmpi-81115e31e2-worker-1         1/1     Running   0          4m14s
d4p-openmpi-bd35283fe8-launcher-q44ft   1/1     Running   0          4m4s
d4p-openmpi-bd35283fe8-worker-0         1/1     Running   0          4m10s
d4p-openmpi-bd35283fe8-worker-1         1/1     Running   0          4m10s
d4p-openmpi-ecb48a1cc9-launcher-qxwr8   1/1     Running   0          4m30s
d4p-openmpi-ecb48a1cc9-worker-0         1/1     Running   0          4m34s
d4p-openmpi-ecb48a1cc9-worker-1         1/1     Running   0          4m34s
d4p-openmpi-f60afa4989-launcher-ndbjm   1/1     Running   0          4m22s
d4p-openmpi-f60afa4989-worker-0         1/1     Running   0          4m26s
d4p-openmpi-f60afa4989-worker-1         1/1     Running   0          4m26s
d4p-registry-5cdd75879-hl5h5            1/1     Running   0          4h37m
d4p-registry-db-78587bdc49-7rb8h        1/1     Running   0          4h37m
dare-login-5fc84cf967-wgkwr             1/1     Running   0          4h37m
```

5. After plots are computed, resulting plotfiles and referring logfiles are available

```
Concentration_Info_01_Jan_2017_at_05-30.png  Deposit_thickness_01_Jan_2017_at_06-00.png  Height_Averaged_Conc_01_Jan_2017_at_00-00.png
Concentration_Info_01_Jan_2017_at_06-00.png  Ground_load_01_Jan_2017_at_00-00.png        Height_Averaged_Conc_01_Jan_2017_at_00-30.png
Deposit_thickness_01_Jan_2017_at_00-00.png   Ground_load_01_Jan_2017_at_00-30.png        Height_Averaged_Conc_01_Jan_2017_at_01-00.png
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user3/runs/user3_20200717151
user3_20200717151117_bf3f46e0a6704ea0ae4ac66b7f89121f/ user3_20200717152720_499a45bf79fb43a4beef8752dde57772/
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user3/runs/user3_20200717152720_499a45bf79fb43a4beef8752dde57772/output/
Concentration_Info_01_Jan_2017_at_00-00.png  Deposit_thickness_01_Jan_2017_at_00-30.png  Ground_load_01_Jan_2017_at_01-00.png         Height_Averaged_Conc_01_Jan_2017_at_01-30.
Concentration_Info_01_Jan_2017_at_00-30.png  Deposit_thickness_01_Jan_2017_at_01-00.png  Ground_load_01_Jan_2017_at_01-30.png         Height_Averaged_Conc_01_Jan_2017_at_02-00.
Concentration_Info_01_Jan_2017_at_01-00.png  Deposit_thickness_01_Jan_2017_at_01-30.png  Ground_load_01_Jan_2017_at_02-00.png         Height_Averaged_Conc_01_Jan_2017_at_02-30.
Concentration_Info_01_Jan_2017_at_01-30.png  Deposit_thickness_01_Jan_2017_at_02-00.png  Ground_load_01_Jan_2017_at_02-30.png         Height_Averaged_Conc_01_Jan_2017_at_03-00.
Concentration_Info_01_Jan_2017_at_02-00.png  Deposit_thickness_01_Jan_2017_at_02-30.png  Ground_load_01_Jan_2017_at_03-00.png         Height_Averaged_Conc_01_Jan_2017_at_03-30.
Concentration_Info_01_Jan_2017_at_02-30.png  Deposit_thickness_01_Jan_2017_at_03-00.png  Ground_load_01_Jan_2017_at_03-30.png         Height_Averaged_Conc_01_Jan_2017_at_04-00.
Concentration_Info_01_Jan_2017_at_03-00.png  Deposit_thickness_01_Jan_2017_at_03-30.png  Ground_load_01_Jan_2017_at_04-00.png         Height_Averaged_Conc_01_Jan_2017_at_04-30.
Concentration_Info_01_Jan_2017_at_03-30.png  Deposit_thickness_01_Jan_2017_at_04-00.png  Ground_load_01_Jan_2017_at_04-30.png         Height_Averaged_Conc_01_Jan_2017_at_05-00.
Concentration_Info_01_Jan_2017_at_04-00.png  Deposit_thickness_01_Jan_2017_at_04-30.png  Ground_load_01_Jan_2017_at_05-00.png         Height_Averaged_Conc_01_Jan_2017_at_05-30.
Concentration_Info_01_Jan_2017_at_04-30.png  Deposit_thickness_01_Jan_2017_at_05-00.png  Ground_load_01_Jan_2017_at_05-30.png         Height_Averaged_Conc_01_Jan_2017_at_06-00.
Concentration_Info_01_Jan_2017_at_05-00.png  Deposit_thickness_01_Jan_2017_at_05-30.png  Ground_load_01_Jan_2017_at_06-00.png         logs.txt
Concentration_Info_01_Jan_2017_at_05-30.png  Deposit_thickness_01_Jan_2017_at_06-00.png  Height_Averaged_Conc_01_Jan_2017_at_00-00.png
Concentration_Info_01_Jan_2017_at_06-00.png  Ground_load_01_Jan_2017_at_00-00.png        Height_Averaged_Conc_01_Jan_2017_at_00-30.png
Deposit_thickness_01_Jan_2017_at_00-00.png   Ground_load_01_Jan_2017_at_00-30.png        Height_Averaged_Conc_01_Jan_2017_at_01-00.png
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user4/runs/user4_2020071715
user4_20200717151119_04b7cda17d4546d8bdf37448a82fa374/ user4_20200717152723_2a893f9fe3564b12aeca03f70751e4ce/
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user4/runs/user4_20200717152723_2a893f9fe3564b12aeca03f70751e4ce/output/
Concentration_Info_01_Jan_2017_at_00-00.png  Deposit_thickness_01_Jan_2017_at_00-30.png  Ground_load_01_Jan_2017_at_01-00.png         Height_Averaged_Conc_01_Jan_2017_at_01-30.
Concentration_Info_01_Jan_2017_at_00-30.png  Deposit_thickness_01_Jan_2017_at_01-00.png  Ground_load_01_Jan_2017_at_01-30.png         Height_Averaged_Conc_01_Jan_2017_at_02-00.
Concentration_Info_01_Jan_2017_at_01-00.png  Deposit_thickness_01_Jan_2017_at_01-30.png  Ground_load_01_Jan_2017_at_02-00.png         Height_Averaged_Conc_01_Jan_2017_at_02-30.
Concentration_Info_01_Jan_2017_at_01-30.png  Deposit_thickness_01_Jan_2017_at_02-00.png  Ground_load_01_Jan_2017_at_02-30.png         Height_Averaged_Conc_01_Jan_2017_at_03-00.
Concentration_Info_01_Jan_2017_at_02-00.png  Deposit_thickness_01_Jan_2017_at_02-30.png  Ground_load_01_Jan_2017_at_03-00.png         Height_Averaged_Conc_01_Jan_2017_at_03-30.
Concentration_Info_01_Jan_2017_at_02-30.png  Deposit_thickness_01_Jan_2017_at_03-00.png  Ground_load_01_Jan_2017_at_03-30.png         Height_Averaged_Conc_01_Jan_2017_at_04-00.
Concentration_Info_01_Jan_2017_at_03-00.png  Deposit_thickness_01_Jan_2017_at_03-30.png  Ground_load_01_Jan_2017_at_04-00.png         Height_Averaged_Conc_01_Jan_2017_at_04-30.
Concentration_Info_01_Jan_2017_at_03-30.png  Deposit_thickness_01_Jan_2017_at_04-00.png  Ground_load_01_Jan_2017_at_04-30.png         Height_Averaged_Conc_01_Jan_2017_at_05-00.
Concentration_Info_01_Jan_2017_at_04-00.png  Deposit_thickness_01_Jan_2017_at_04-30.png  Ground_load_01_Jan_2017_at_05-00.png         Height_Averaged_Conc_01_Jan_2017_at_05-30.
Concentration_Info_01_Jan_2017_at_04-30.png  Deposit_thickness_01_Jan_2017_at_05-00.png  Ground_load_01_Jan_2017_at_05-30.png         Height_Averaged_Conc_01_Jan_2017_at_06-00.
Concentration_Info_01_Jan_2017_at_05-00.png  Deposit_thickness_01_Jan_2017_at_05-30.png  Ground_load_01_Jan_2017_at_06-00.png         logs.txt
Concentration_Info_01_Jan_2017_at_05-30.png  Deposit_thickness_01_Jan_2017_at_06-00.png  Height_Averaged_Conc_01_Jan_2017_at_00-00.png
Concentration_Info_01_Jan_2017_at_06-00.png  Ground_load_01_Jan_2017_at_00-00.png        Height_Averaged_Conc_01_Jan_2017_at_00-30.png
Deposit_thickness_01_Jan_2017_at_00-00.png   Ground_load_01_Jan_2017_at_00-30.png        Height_Averaged_Conc_01_Jan_2017_at_01-00.png
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user5/runs/user5_2020071715
user5_20200717151123_78935baf28ba438a8e8b6ca0d29095ff/ user5_20200717152725_64726554a3b649c3ab455403017e023c/
root@exec-api-7fd7bdbff4-c8dbz:/home/mpiuser/sfs# ls user5/runs/user5_20200717152725_64726554a3b649c3ab455403017e023c/output/
Concentration_Info_01_Jan_2017_at_00-00.png  Deposit_thickness_01_Jan_2017_at_00-30.png  Ground_load_01_Jan_2017_at_01-00.png         Height_Averaged_Conc_01_Jan_2017_at_01-30.
Concentration_Info_01_Jan_2017_at_00-30.png  Deposit_thickness_01_Jan_2017_at_01-00.png  Ground_load_01_Jan_2017_at_01-30.png         Height_Averaged_Conc_01_Jan_2017_at_02-00.
```

To check the capacity utilization of the DARE system and observe the requirements of the individual components of the platform, especially during specific test runs and trainings, partners kept an eye on the monitoring output.

Figure 16 shows the CPU usage of different platform components, recorded during the week after the webinar: Introduction to the DARE platform - Focusing on Climate and Atmospheric Sciences (October 16, 2020), where all participating users had access to the system and were able to test the DARE applications.

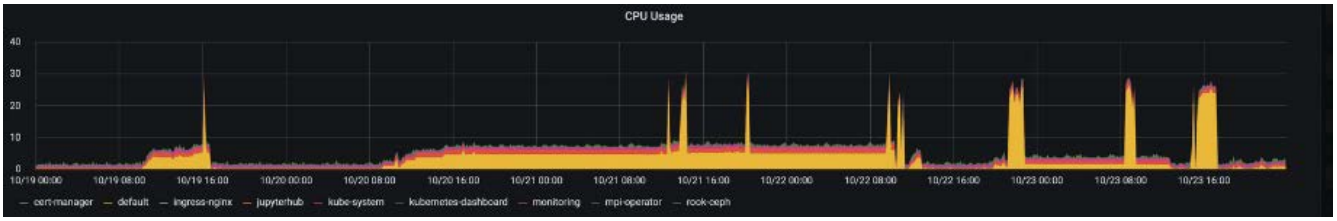

Figure 16: CPU usage during DARE training event

Figure 17 shows the CPU and memory usage, during the run of different SPECFEM3D simulations (seismology test case, Deliverable D6.4) with up to 24 cores. It is noticeable that during these simulations the CPU usage temporally exceeds the usage of 30 CPUs, which indicates CPU consumption by platform processes.
Memory consumptions by a heavy numerical simulation like SPECFEM3D seems to be adequate compared to a run in a run on bare metal.

Figure 17: CPU/Memory usage during specfem simulations

Relatively few resources are needed by the DARE components. In figure 18, you can see that during the deployment process as expected the exec-api starting all Kubernetes jobs needs most CPU power and is running on more than one core (here 3).

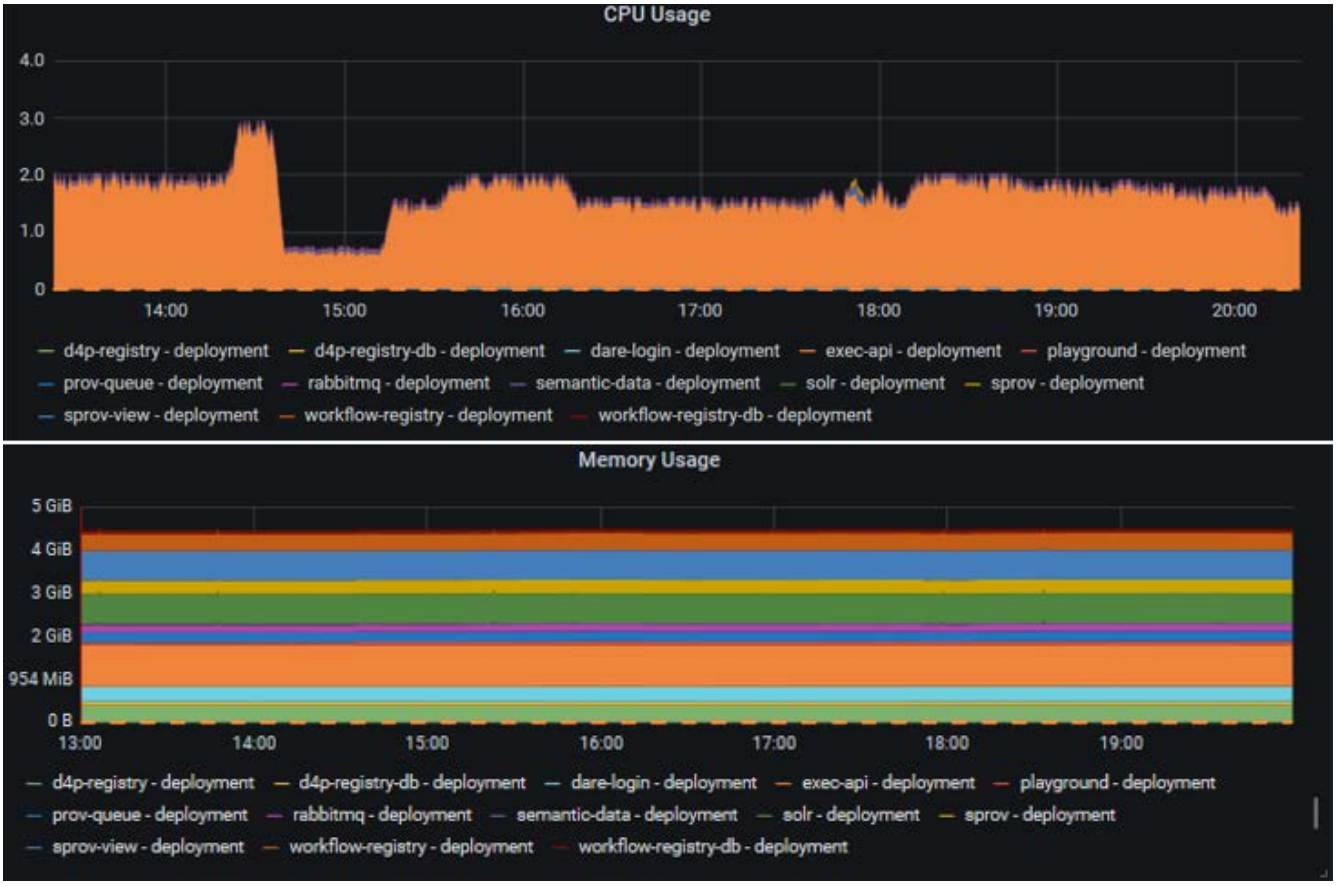

Figure18: CPU/Memory usage of DARE components during deployment

In the stateful mode the CPU usage of the Kubernetes services are marginal. Interesting for us was the minimal resource consumption of the DARE sprov-db provenance service during an application run like SPECFEM3D.

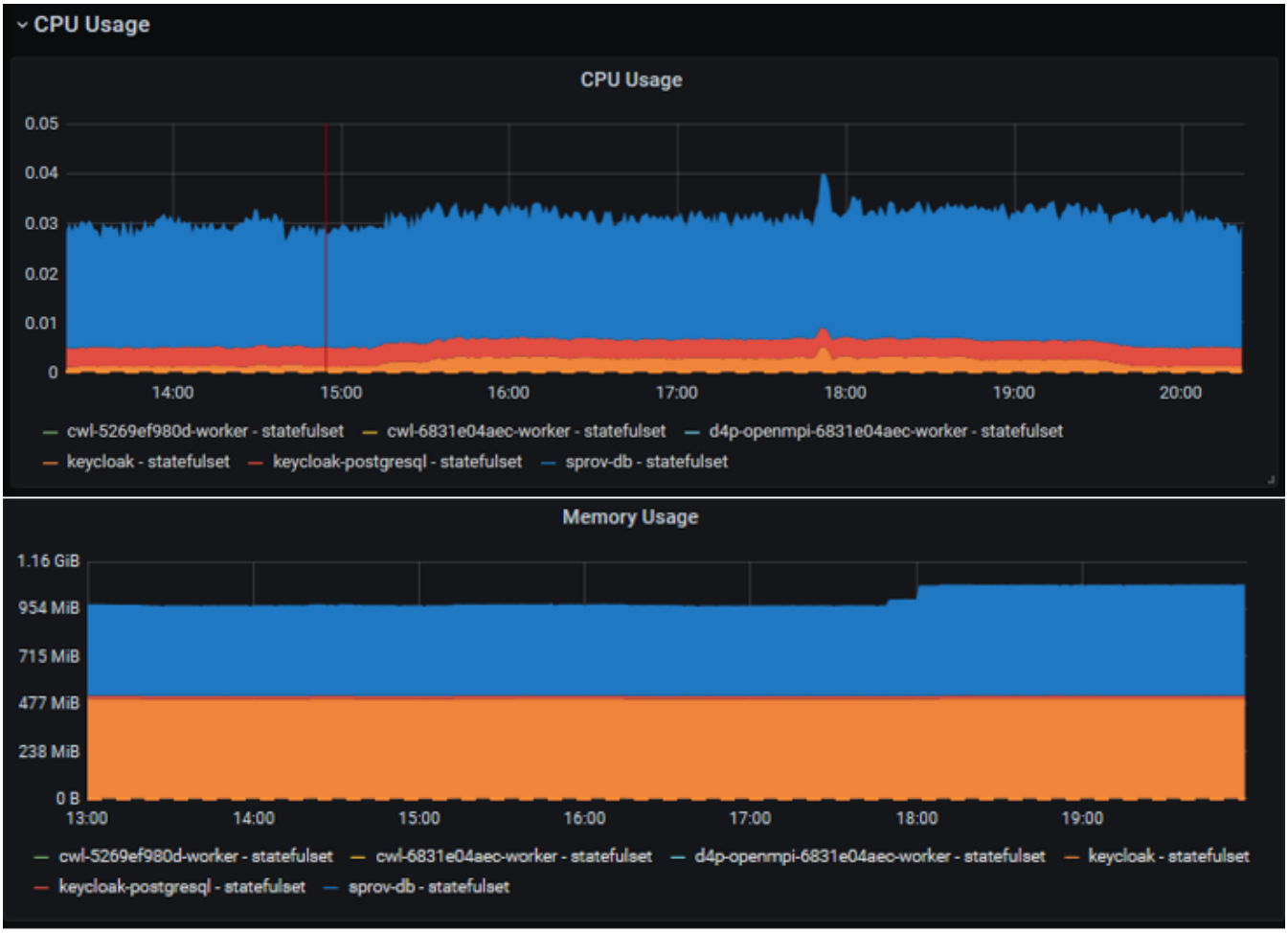

Figure 19: CPU/Memory usage of stateful DARE components

Overall, it can be concluded that the system's resource consumption is low. Even the Provenance service steals nearly nothing from the actual application. Furthermore, the virtualized infrastructure with integrated virtual shared storage runs stable and efficient, and even in multi-user operations it runs with a reasonable resource consumption. Since the infrastructure stack is cloud native, it can be used like a scalable, virtual HPC system even allowing the run for large MPI applications.

6 Conclusion

In this deliverable, we reported on the work performed in the scope of WP5, which was responsible for the provision and management of the cloud environments made available for the DARE platform.

To this extent, we initially described the cloud-ready DARE platform stack focusing on the infrastructural components and tools like e.g., the AA mechanism and gave a rough description of the cloud resources mobilized for DARE and provisioned at SCAI and GRNET. With the decision to rely on an architecture based on orchestrated micro services, DARE aimed to develop a packaged development kit easily deployable on cloud resources especially having in mind the EOSC. Moreover, we described the platform deployment strategy followed by the project and summarized the public releases and deployments. Finally, we described the main usage purposes of the DARE environments tailored to the needs of the climate and seismology use cases, and the additional volcanology test case, and presented some monitoring and testing results of the operational DARE environment provided at SCAI. This demonstrated that the provided platform is stable, efficient and performs with reasonable resource consumption.

In retrospect, it was a good decision to follow the above described strategies and to provide two separate platform environments. Besides the fact that the rapid provision of the testbed gave the developers access to a development environment at an early stage in the project, this approach also facilitated the co-development of the individual components. While partners from the use cases, were already able in the testbed environment to try out and work with newly developed and not completely stable beta-features of the DARE stack, others were able to rely on a stabile variant of the DARE platform to optimise their workflows or to perform trainings.

Also the decision to rely on orchestrated micro services facilitated the independent development of the needed individual components in DARE. However, one should not underestimate that a high level of communication between all partners was required to make sure that all components cooperate effectively and to ensure that an update to one service didn’t break some others functionality. For future projects, it therefore seems advisable to place even greater emphasis on testing the individual operating components and their interaction.

One of the particularly challenging parts in WP5 was e.g. to meet the sometimes contradictory requirements of a completely integrated platform (operational environment at SCAI) and those of a desired locally installable environment. For example, in connection with the choice and integration of an appropriate AA solution. Also the extremely fast-changing universe around containers and orchestration was quite challenging and led to the need of reworking developed solutions. In some cases solutions developed in DARE even became superfluous.