**H2020-EINFRA-2017**

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

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

# D5.4 Operational Requirements and Guidelines II

| | |
|---|---|
| **Project Reference No** | 777413 — DARE — H2020-EINFRA-2017 / EINFRA-21-2017 |
| **Deliverable** | D5.4 Operational Requirements and Guidelines II |
| **Work package** | WP5: Platform Operation and Maintenance |
| **Tasks involved** | T5.1: Provision of Relevant Cloud Infrastructure<br>T5.2: Provision of Pre-release Testbeds<br>T5.3: Deployment Strategy and Platform Operation<br>T5.4 Operational Requirements Specifications and Roadmap |
| **Type** | R: Document, report |
| **Dissemination Level** | PU = Public |
| **Due Date** | 31/12/2020 |
| **Submission Date** | 30/12/2020 |
| **Status** | Draft |
| **Editor(s)** | Stavros Sachtouris (GRNET) |
| **Contributor(s)** | André Gemünd (SCAI) |
| **Reviewer(s)** | Alessandro Spinuso (KNMI) |

| | |
|---|---|
| **Document description** | This deliverable reports on the set of guidelines and best practices for managing and maintaining a DARE platform deployment on the cloud, regarding infrastructure, A/A, monitoring, security and recovery issues as well operational guidelines. It is a follow up to D5.3, the focus of which was on providing suggestions for developing the platform and experimenting with available solutions as DARE is being built. The present document provides guidelines and instructions for the operators, developers and scientific domain experts on how to operate a functional and tested platform. |

## Document Revision History

| Version | Date | Modifications Introduced | |
|---|---|---|---|
| | | Modification Reason | Modified by |
| **1** | 31/10/2020 | Initial Structure | S. Sachtouris (GRNET) |
| **2** | 14/12/2020 | Context and generic info on each chapter | S. Sachtouris (GRNET) |
| **3** | 15/12/2020 | Production specifics | A. Gemünd (SCAI) |
| **4** | 17/12/2020 | Infrastructure, Capacity | S. Sachtouris (GRNET) |
| **5** | 19/12/2020 | A/A, Monitoring | S. Sachtouris (GRNET) |
| **6** | 20/12/2020 | First draft | S. Sachtouris (GRNET) |
| **7** | 22/12/2020 | Second Draft | S. Sachtouris (GRNET) |
| **8** | 23/12/2020 | Finalized | S. Sachtouris (GRNET) |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

## Executive Summary

This deliverable reports on the set of guidelines and best practices for managing and maintaining a DARE platform deployment on premises or on the cloud. Specifically, it contains installation, configuration and maintenance instructions and best practices regarding infrastructure, container management, networking, storage, scaling, authentication, security and disaster recovery, as foreshadowed in D5.3[1]. This information is derived from reliable vendor resources as well as the experience of developing, maintaining and using the testbed and production platforms described in D5.2[2].

[1] D5.3 Operational Requirements and Guidelines I, http://project-dare.eu/wp-content/uploads/2019/03/D5.3-Operational-Requirements-and-Guidelines-I_final_draft.pdf

[2] D5.2 Platform Infrastructure, Usage & Deployment II

## Table of Contents

## List of Terms and Abbreviations

| Abbreviation | Definition |
|---|---|
| DARE | Delivering Agile Research Environments |
| K8s | Kubernetes |
| IaaS | Infrastructure as a Service |
| WaaS | Workflow as a Service |
| DRB | Dare-specific Knowledge Base |
| P4 | Protected Pervasive Persistent Provenance |
| MPI | Message Passing Interface |
| PE | Process Element |
| CWL | Common Workflow Language |
| GKE | Google K8s Engine |
| EKS | Amazon Elastic K8s Services |
| HPC | High Performance Computing |
| VM | Virtual Machine |
| LTS | Long Term Support |
| TLS | Transport Layer Security |
| SSH | Secure Shell |
| CLI | Command Line Interface |
| AWS | Amazon Web Services |
| OS | Operating System |
| KVM | Kernel-based Virtual Machine |
| ACME | Automated Certificate Management Environment |
| SSO | Single Sign On |
| RBD | RADOS Block Device |
| PV | Persistent Volume |
| PVC | Persistent Volume Claim |
| EOSC | European Open Science Cloud |
| EGI | European Grid Initiative |
| EUDAT | European Association of Databases for Education and Training |
| PPK | Private-Public Key |
| SAML | Single Assertion Markup Language |
| RBAX | Role-Based Access Control |
| DB | Database |

# 1 Introduction

This document is intended as a collection of instructions, guidelines and suggestions for deploying and operating the DARE platform. It must be thought of as the continuation of D5.3 ("Operational Requirements and Guidelines I") which was composed at the beginning of the project, when the platform was still in the planning face, while D5.4 is delivered at the end of the DARE project. In the former, information was acquired mostly from sources related to industry standards on cloud computing, container management, etc., while in the later most of the information was collected while developing and trying out the platform.

The DARE platform sprang out from the collaboration between experts in diverse fields i.e., Computer Science, Seismology and Meteorology. Since its conception it was designed as a distributed platform, consisting of containerized applications and services, living on the cloud. Its target users are scientific communities in need of processing high loads of structured data, while storing provenance information in a way meaningful to these communities and usable by future processes.

Most suggestions and guidelines from D5.3 acted as the basis for building the platform. In D5.3 it is proposed to follow a cloud-native approach, which was adopted by DARE designers and developers to a great extent. In specific, it was proposed to rely on containers for handling the auxiliary services, as well as to support novel services and to run processes. Kubernetes (K8s)[3] was elected as the most suitable candidate from a selection of mature and industry-hardened container management technologies, the Prometheus[4]-Alertmanager[5]-Grafana[6] stack was chosen for monitoring, tools like kubespray[7] and minikube[8] were often used to spawn DARE instances for experimentation and development, while the adopted networking solution (Calico[9]) was also proposed there. What's more, the platform was built with an IaaS or a K8s-ready cluster in mind, as well as in a way that conforms with EOSC standards and tooling.

There are some aspects, though, where the guidelines were not followed. For instance, Keycloak[10] was selected as the authentication/authorization (A/A) handler, and Rook CEPH[11] as the storage layer, since they satisfied the requirements and needs of the platform. The user and role policies constitute the most notable diversion. The initial suggestion was an elaborate but complicated scheme, aimed primarily to support the requirements of a public service. After it was suggested to the scientific communities, though, their feedback directed us towards an approach of lower complexity. As the project evolved, it became apparent that the platform was primarily targeted to scientific communities and the institutions that foster them, thus it was necessary to simplify access and minimize operational load so that DARE would become an attractive solution to them.

[3] https://kubernetes.io/

[4] https://www.prometheus.io/

[5] https://github.com/prometheus/alertmanager

[6] https://grafana.com/

[7] https://kubespray.io/

[8] https://minikube.sigs.k8s.io/

[9] https://kubernetes.io/docs/tasks/administer-cluster/network-policy-provider/calico-network-policy/

[10] https://www.keycloak.org/

[11] https://rook.io/docs/rook/v1.5/ceph-storage.html

At the beginning of the project, it was unclear whether the platform would evolve to be (a) a service maintained by a team of informational technology experts and offered as a service to a wide range of scientific federations, or (b) a tool to be installed and maintained on premises of various institutions and organizations. In the following, when we refer to DARE installations and in order to distinguish between the two cases, we use the terms “public installation” and “local installation” respectively. DARE focuses primarily on local installations, but public ones are supported as well.

## 1.1 Platform architecture overview

From a functional perspective, the DARE platform is build over three pillars[12], namely a Workflow as a Service (WaaS) functionality, a Dare-specific Knowledge Base (DRB) and a set of Provenance tools (Protected Pervasive Persistent Provenance or P4). Each of these groups of functions is provided by software and services that are either adopted by or developed for DARE.

From a systemic perspective, the platform consists of software components bundled in containers, living in a Kubernetes (K8s) cluster. The decision to rely on K8s for the orchestration, management and operations of all DARE components was proposed and taken early[13], so that most of the system design, development process, testing and training could be based on safe assumptions regarding the underlying infrastructure interface and offerings.

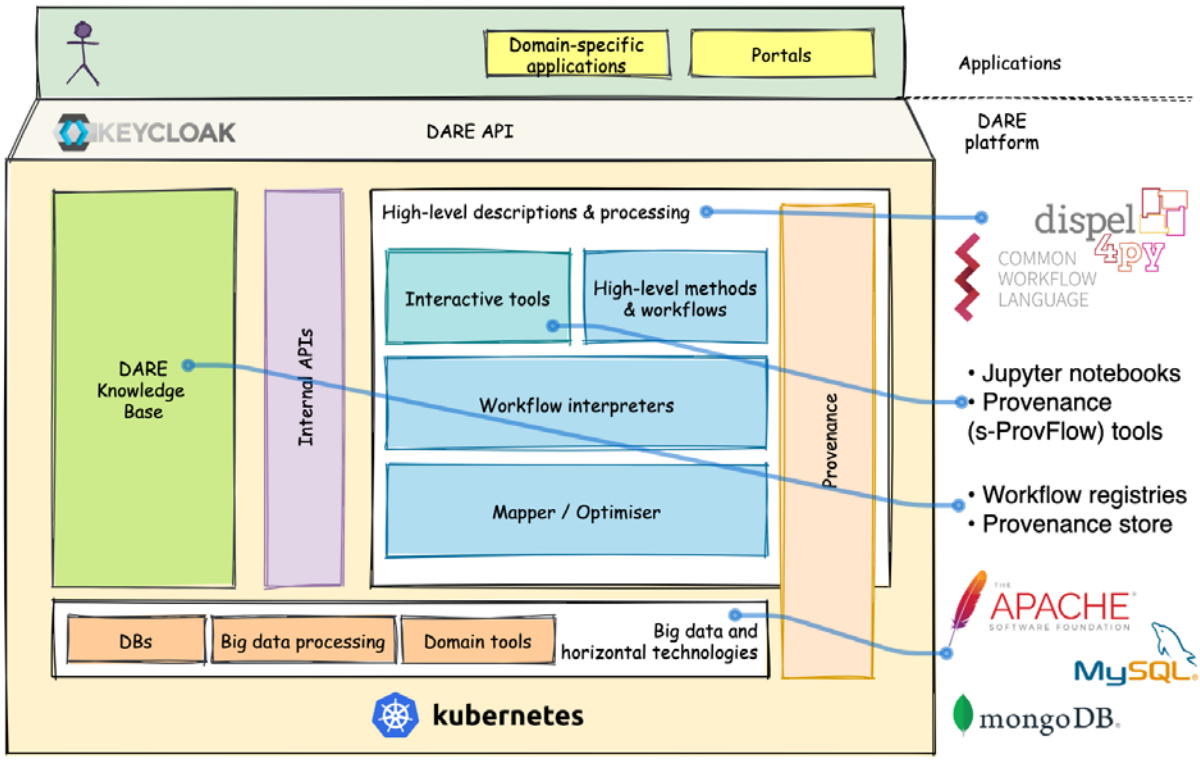


Figure 1: *The structure and main components of the current DARE platform*[14]

[12] See D2.2 “DARE Architecture and Technology Report”, chapter 4 “Architecture and Implementation”

[13] E.g., D5.3 “Operational Requirements and Guidelines II”

[14] From D2.2 “DARE Architecture and Technology Report”, chapter 4 “Architecture and Implementation”

In the following we group the components of the platform in two groups, namely the "DARE components" and the "Platform components". The term "DARE components" refers to software used to process and store data, metadata, provenance information or the results of this process, which is either developed in the scope of the project or configured to serve the aforementioned needs e.g., for workflow execution, indexing, searching etc. The term "Platform components" refers to software used to build the supporting infrastructure e.g. for container management, monitoring, networking, storage, security, availability etc. The DARE components can be further grouped into management components (APIs, provenance, registries) and execution environments (workflows and applications).

A detailed list and description of the DARE components can be found in D2.2[15].The platform adopts and features a variety of well known tools for scientific data processing, indexing, searching etc., like JupyterHub[16], SemaGrow[17] or Exareme[18]. A lot of tools have been developed in the context of the project. The ones summarized below are picked for their significance in the platform ecosystem, and also to demonstrate the diversity of the systemic requirements.

DARE architecture relies on the Message Passing Interface (MPI) protocol (the MPI operator being a platform component) and dispel4py (a DARE component) which relies on MPI for implementing workflow execution. MPI is a reliable and well known parallelization interface which can be configured to utilize a group of connected nodes in the service of a workflow operation. Dispel4py maps graphs of Process Elements (PE) to MPI in order to schedule them on individual nodes. The PEs are stored in a separate registry (dispel4py-registry and CWL[19] workflow registry). An MPI network is being built on top of the underlying network infrastructure when needed, so that it can be ported from and to other platforms. A feature of great value is a provenance functionality, consisting of s-ProvFlow. S-ProvFlow is developed in the context of DARE and features the acquisition, storage, exploration and visualization of provenance data produced at run-time by DARE processes.

[15] D2.2 DARE Architecture and Technology report

[16] https://jupyter.org/hub

[17] http://semagrow.github.io/

[18] http://madgik.github.io/exareme/

[19] https://www.commonwl.org/

# 2 Infrastructure provisioning

The DARE platform relies on a K8s cluster, which can be deployed over various cloud offerings i.e., IaaS systems (like GRNET's Synnefo[20]-powered okeanos-knossos[21], or SCAI's OpenStack[22] cluster) or cloud-based K8s solutions (e.g., Google Kubernetes Engine/GKE[23], Amazon Elastic Kubernetes Service/EKS[24]). The first solution allows institutions to deploy and maintain their own version of the platform on their own premises, given that they have access on an IaaS cloud. The advantages of an on-premises solution is full control over sensitive data as well as the ability to utilize computational and data resources (e.g., HPC clusters, knowledge bases, etc.) available internally to the institution. The advantage of a cloud-based cluster is the simplification of the deployment process and the lower maintenance cost (backup, updates, support, etc.).

The term "infrastructure provisioning" includes the physical or cloud resource allocation as well as the installation of the underlying platform (K8s) and the Platform components. Although from a deployment perspective, Platform and DARE components share a lot of similarities; the both live in containers managed by K8s. From a functional perspective, though, Platform components constitute the scaffolding holding the DARE components into place and allowing them to form a meaningful and usable service. For this reason, we consider Platform components as part of the infrastructure, along with the physical layer and the K8s installation.

## 2.1 IaaS provisioning

Provisioning an IaaS for DARE equals provisioning for the underlying K8s cluster and the MPI network. A K8s cluster requires at least one host as a master. A host is either a physical server or a VM, with the following minimum requirements:

- K8s master: 1.5 core, 2GB RAM
- K8s node: 0.7 cores, 1GB RAM

The resource allocation can be automated with specialized tools. The choice and configuration of tools depends heavily on the special characteristics of the underlying IaaS. The first step is the allocation of a few nodes with the desired characteristics, a secure access method for the operators, floating IPs and a common network. It is suggested to keep record of the process e.g., in a script used to allocate the resources. Practices and tooling for IaaS have been adequately covered in D5.3.

DARE was developed on two IaaS clusters: a Synnefo cluster (okeanos-knossos) offered by GRNET[25] as a testbed and an OpenStack cluster offered by SCAI[26] as a production environment. Both systems provide compute, network and storage resources in the form of virtual machines (VMs), which were provisioned with Terraform[27] in the case of OpenStack, or with kamaki-ansible-role[28] in the case of

[20] https://synnefo.org

[21] https://okeanos-knossos.grnet.g

[22] https://www.openstack.org/

[23] https://cloud.google.com/kubernetes-engine/

[24] https://aws.amazon.com/eks/

[25] See D5.2, chapter 3.2

[26] See D2.2, chapter 3.3

[27] https://www.terraform.io/

[28] https://github.com/saxtouri/kamaki-ansible-role/

Synnefo. The testbed was provisioned with relatively low resources (3 VMs of 16 cores, 16GB RAM and 70GB storage) with more available if needed. The production environment was initially deployed on similar resources which were scaled up when there was demand (e.g., training sessions).

In the following we showcase the provision of infrastructure with automation tools fit for specific IaaS services. DARE has been tested on hosts running Ubuntu 16.04 LTS[29], connected on the same network.

### Terraform and OpenStack

Terraform[30] is a provisioning tools that works well with various IaaS systems, including OpenStack[31]. The following terraform script allocates and builds the infrastructure for a small but functional K8s cluster. The three VMs are declared towards the end.

Terraform script example to provision an OpenStack Cluster for K8s

```
# Configure the OpenStack Provider provider "openstack"
{
  user_name = "admin"
  tenant_name = "admin"
  password = "pwd"
  auth_url = "http://example.org:5000/v2.0"
  region = "RegionOne"
}

# Ubuntu 16.04 LTS as default
variable "image" {
  default = "Ubuntu 16.04"
}

# Medium-sized VM as default
variable "flavor" {
  default = "m1.medium"
}

# Use this private key as default (injected in VM on build)
variable "ssh_key_file" {
  default = "~/.ssh/id_rsa.dare"
}

# Username as default
variable "ssh_user_name" {
  default = "ubuntu"
}

# Setup private network
resource "openstack_networking_subnet_v2" "dare_net" {
  name = "dare_net"
  network_id = "${openstack_networking_network_v2.terraform.id}"
  cidr = "10.0.0.0/24"
  ip_version = 4
  dns_nameservers = ["1.2.3.4","5.6.7.8"]
}
```

[29] http://releases.ubuntu.com/16.04/

[30] https://www.hashicorp.com/products/terraform

[31] https://registry.terraform.io/providers/terraform-provider-openstack/openstack/latest/docs

```
resource "openstack_networking_router_v2" "dare_net" {
  name = "dare_net"
  admin_state_up = "true"
}

resource "openstack_networking_router_interface_v2" "terraform" {
  router_id = "${openstack_networking_router_v2.terraform.id}"
  subnet_id = "${openstack_networking_subnet_v2.terraform.id}"
}

resource "openstack_compute_floatingip_v2" "terraform" {
  pool = "${var.pool}"
  depends_on = ["openstack_networking_router_interface_v2.terraform"]
}

# Allocate 3 equal VMs
resource "openstack_compute_instance_v2" "kubernetes_1" {
  name      = "kubernetes_1″
  image_name = "${var.image}"
  flavor_name = "${var.flavor}"
  key_pair = "${openstack_compute_keypair_v2.terraform.name}"
  floating_ip = "${openstack_compute_floatingip_v2.terraform.address}"
  network {
    uuid = "${openstack_networking_network_v2.terraform.id}"
  }

  provisioner "remote-exec" {
    connection {
    user = "${var.ssh_user_name}"
    key_file = "${var.ssh_key_file}"
  }
}

resource "openstack_compute_instance_v2" "kubernetes_2" {
  name      = "kubernetes_2″
  …
}

resource "openstack_compute_instance_v2" "kubernetes_3" {
  name      = "kubernetes_3″
  …
}
```

Note how access to VMs is controlled with SSH. This key is different to the TLS keys used by K8s to securely interconnect its parts. It is, instead, the master-key that gives superuser access on the infrastructure. In this case, the "ubuntu" user of each VM is sudo-priviledged.

### Synnefo with kamaki and ansible

On Synnefo clusters, terraform would not work out of the box, although some scripts could work with some adjustments. It is, instead, suggested to rely on kamaki[32], a Command Line Interface (CLI) and python library for Synnefo, combined with ansible[33], using the kamaki-ansible-role[34], which was developed in the context of DARE.

[32] https://github.com/grnet/kamaki

[33] https://www.ansible.com/

[34] https://github.com/saxtouri/kamaki-ansible-role

Here is an ansible playbook to prepare the same resources as the ones in the Terraform script in “Terraform and Openstack” subsection above.

```
tasks:
- name: Provision DARE infra
  hosts: localhost
  tasks:
    - import_role:
        name: kamaki-ansibe-role
    - name: Authenticate cloud
      cloud:
        url='https://example.org/identity/v2.0'
        token='MY-SYNNEFO-TOKEN'
        project_id='MY-PROJECT'
      register: cloud
   - name: Create PPK
      keypair:
        cloud={{ cloud }}
        name='My keypair'
      register: ppk
    - name: Save private key
      copy:
        # Only if this is a new key
        content={{ ppk.keypair.private_key }}
        dest=~/.ssh/id_rsa.dare
      register: saved_key
   - name: Create private network
      network:
        cloud={{ cloud }}
        name='dare_net'
        dhcp=True
        cidr='10.0.0.0/24'
      register: dare_net
   - name: Create IP
      public_ip:
        cloud={{ cloud }}
      register: ip
   - name: Create Kubernetes 1
      server:
        cloud={{ cloud }}
        name='Kubernetes_1'
        flavor_id=260
        image_id='051669a1-835a-4e01-995e-1d21c74839c7'
        public_ip={{ ip }}
        network={{ dare_net }}
        keypair={{ ppk }}
    register: kubernetes_1
   - name: Create Kubernetes 2
      server:
        ...
   - name: Create Kubernetes 3
      server:
        ...
```

Note that flavor_id and image_id are selected by the user using the Synnefo mechanisms for listing available profiles to correspond to a medium-sized machine (flavor) running Ubuntu 16.04 LTS (image).

### Post-build provision

Nodes must support containers. In DARE we opted for Docker[35] as the underlying container framework. The preparation of a node for containers depends on the virtualization setup and the chosen OS image, but the vast majority of IaaS offerings fully support containers. Details on how to prepare a system for Docker can be found in official Docker documentation[36].

Trivial post-build operations (e.g., OS updates) should also be applied at this point. It is advised for these operations to be included in the original provisioning script.

In our example (Ubuntu 16.04) this is the process:

```
# Update system
sudo apt update && sudo apt upgrade -y
# Install docker
sudo apt install docker.io
# Enable docker
sudo systemctl enable docker
```

## 2.2 Setting up Kubernetes

Kubernetes has become an industry standard since the submission of D5.3, two years ago. Still, the architecture details and tools of that document are still relevant. Since then, K8s was stabilized and its tools became trustworthy. As a consequence, auxiliary tooling used to setup and configure kubernetes is not always needed.

In principle, any standard K8s cluster could support the DARE platform, as long as enough resources are available, interconnected and ready to support containers. There are various tools to deploy Kubernetes on such a system, the most mature of which have been surveyed in D5.3. In the context of DARE, we have relied from time to time on kubespray, minikube and, finally, the tools shipped with K8s itself (kubeadm, kubectl). As the DARE platform became more stable, so did the built-in K8s tools, which are the currently suggested approach for setting up k8s.

Kubespray is a collection of configuration templates and tools organized in ansible playbooks, provisioning tools and domain knowledge for generic OS/Kubernetes clusters configuration management tasks. It can connect to the most popular commercial IaaS offerings (e.g., AWS, Azure) as well as OpenStack clusters and it supports Terraform as an infrastructure provisioner.

Minikube is a turn-key solution for setting up K8s locally with a single command. It can run on a Docker-ready host or on virtualization technologies (e.g., KVM[37], VirtualBox[38], etc.). It was useful on testbed as an easy way to start a cluster for interim installations, as well for local experiments. After the initial setup, additional resources had to be added to the cluster with standard K8s tools (kubeadm or kubectl).

[35] https://www.docker.com/

[36] https://docs.docker.com/get-docker/

[37] https://www.linux-kvm.org/

[38] https://www.virtualbox.org/

We settled for the tools included with the main K8s offering. In the following we showcase the installation of an adequately provisioned cluster like the ones used on testbed, production or the examples in "IaaS provisioning" subsection. The script starts with the installation of the tools themselves, as well as kubelet, which is a mandatory component of a K8s cluster.

Install K8s:

```
# Add Kubernetes signing key
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add

# Add Kubernetes Repository for Ubuntu 16.04 (Xenial)
sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"

# Install Kubeadm
sudo apt install kubeadm=1.15.3-00 kubectl=1.15.3-00 kubelet=1.15.3-00

# Initialize Kubernetes on the master node
sudo kubeadm init

# start using your cluster
mkdir -p $HOME/.kube & sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config & sudo
chown $(id -u):$(id -g) $HOME/.kube/config

# Deploy a Pod Network through the master node
kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64
| tr -d '\n')"

# Run
kubectl taint nodes --all node-role.kubernetes.io/master-
```

A copy of the file from https://cloud.weave.works/k8s/net can be found in the appendix ("K8s Network")

By following the examples presented so far, we end up with a still unconfigured but functional K8s cluster. At this point, operators may find it useful to install kubectl on their machine and use it to control the cluster remotely. After the installation, the file ~/.kube/config has to be copied from the initial cluster node (e.g., "kubernetes_1") to the operators' host.

## 2.3 Setting up Platform Components

Platform components include the networking (Calico) and storage (Rook CEPH) handlers, Routing (NginX Ingress[39]) and Security (cert-manager, Harbor[40]), the A/A mechanism (Keycloak), Database Management Systems (PostgreSQL[41], Mongodb[42], Virtuoso[43]) and index services (SOLR/Lucene[44]) and

[39] https://kubernetes.github.io/ingress-nginx/

[40] https://goharbor.io/

[41] https://www.postgresql.org/

[42] https://www.mongodb.com/

[43] https://github.com/openlink/virtuoso-opensource

[44] https://lucene.apache.org/solr/

MPI support (mpi-operator).

A helpful list of scripts and configurations can be found in the dare platform, so it is a good idea to get a copy of the source from gitlab:

```
$ git clone https://gitlab.com/project-dare/dare-platform.git
```

### Helm/Tiller

All these components are installed and managed on K8s with Helm/Tiller. It can be thought of as a package manager for K8s. Helm is based on yaml files (charts) containing installation details and configuration properties. The tool keeps track of installed and deployed versions and allows the management of upgrades for each individual chart.

For DARE, the version 3.3.1 has to be downloaded from https://github.com/helm/helm/releases and the binary has to be placed manually at the path (e.g., /usr/local/bin), as shown below.

```
## Download, unpack and install release package
$ wget https://get.helm.sh/helm-v3.1.1-linux-amd64.tar.gz
$ tar xf helm-v3.1.1-linux-amd64.tar.gz
$ sudo mv linux-amd64/helm /usr/local/bin/

## create service account
$ kubectl create serviceaccount -n kube-system tiller

## create role for RBAC
$ kubectl create clusterrolebinding tiller-binding --clusterrole=cluster-admin --
serviceaccount kube-system:tiller

## Update package sources
$ helm repo update
```

The last step will likely fail, since there are no helm packages installed yet.

### MPI operator

To support MPI, download version 0.1.0[45] of the MPI operator and unzip it. Edit the file "mpi-operator-0.1.0/deploy/3-mpi-operator.yaml" so that the version of the two images is set to 0.1.0 since the MPI-based executions of the platform require this exact version to function. The resulting find can be found in the appendix ("mpi-operator-0.1.0/deploy/3-mpi-operator.yaml"). The operator can be loaded with "kubectl create".

```
$ for script in mpi-operator-0.1.0/deploy/*.yaml; do
  kubectl create -f $script;
done;
```

[45] https://github.com/kubeflow/mpi-operator/archive/0.1.0.tar.gz

### Rook CEPH

The storage layer is handled with Rook, which can offer a CEPH block storage implementation. The platform depends on running the 0.8 release from the Rook repository[46]. Rook example scripts for CEPH on kubernetes are adequate for our purposes. At the end, we need to set "rook-ceph-block" as default.

```
$ git clone https://github.com/rook/rook.git
$ cd rook && git checkout release-0.8
$ cd cluster/examples/kubernetes/ceph
$ kubectl create -f operator.yaml
$ kubectl create -f cluster.yaml
$ kubectl create -f filesystem.yaml
$ kubectl create -f storageclass.yaml
# Set rook-ceph-block as default storage class
$ kubectl patch storageclass rook-ceph-block -p \
'{"metadata":{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

### NginX Ingress

DARE relies on Calico to handle networking. K8s is instructed to use the Calico plugin at a later stage, when applying the general cluster configuration. At this point, though, we can configure an Ingress service to handle HTTP/S traffic with Ngin.

For that we need to create an "ingress-deployment/nginx-ingress.yaml" file

```
kind: Service
apiVersion: v1
metadata:
  name: ingress-nginx
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
spec:
  externalTrafficPolicy: Local
  type: LoadBalancer
  selector:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
  ports:
    - name: http
      port: 80
      targetPort: http
    - name: https
      port: 443
      targetPort: https
```

We also need to get the mandatory configurations[47] ("mandatory.yaml" in the appendix), which must be applied first:

[46] https://github.com/rook/rook.git

[47] https://raw.githubusercontent.com/kubernetes/ingress-nginx/130af33510882ae62c89277f2ad0baca50e0fafe/deploy/static/mandatory.yaml

```
kubectl apply -f mandatory.yaml
kubectl apply -f ingress-deployment/nginx-ingress.yaml
```

### Cert-manager

The platform security policy requires the issuance and management of multiple X.509[48] certificates for various purposes, most of which regard the internals of K8s. Cert-manager provides this functionality "out of the box", while it is loaded with features necessary for maintaining security as the service grows.

Cert-manager can be configured to support either the "Let's Encrypt" Certification Authority[49] or other certificate authorities (including internal ones) or even self signed certificates. To maintain certificates for the externally reachable services and pages, DARE uses the "Let's Encrypt" through its ACME protocol[50]. The Cert-Manager addon automates this process even further so that certificates are automatically issued, configured in the ingress and updated based on annotations in K8s descriptors. For installation, we use the official Helm package from Helm hub[51].

```
Install certificate manager
```

```
## install custom resource definitions
$ kubectl apply --validate=false -f 00-cards.yaml[52]

## Add the Jetstack Helm repository
$ helm repo add jetstack https://charts.jetstack.io

## Install the cert-manager helm chart
$ helm install cert-manager --version v0.14.0 jetstack/cert-manager

## Install the letsencrypt ClusterIssuer (careful: needs customization!)
$ kubectl apply -f letsencrypt-prod.yaml[53]
```

### Keycloak

Authentication and Authorization is managed by Keycloak, an A/A tool with features like turn-key user management, Single Sign On (SSO) or authentication through federations. The latest version of DARE relies on Keycloak 8.0.0 which is installed with Helm. It is advised to set up Keycloak after cert-manager, in order to ensure secure configuration over HTTPS.
An example of the file "keycloak-values.yaml" can be found in the appendix and also on DARE gitlab repository[54].

```
## Add codecentric package source and update package sources.
$ helm repo add codecentric https://codecentric.github.io/helm-charts
```

[48] https://www.itu.int/rec/T-REC-X.509

[49] https://letsencrypt.org/

[50] https://tools.ietf.org/html/rfc8555

[51] https://hub.helm.sh/charts/jetstack/cert-manager

[52] https://raw.githubusercontent.com/jetstack/cert-manager/release-0.14/deploy/manifests/00-crds.yaml

[53] https://raw.githubusercontent.com/kubernetes/k8s.io/master/cert-manager/letsencrypt-prod.yaml

[54] https://gitlab.com/project-dare/dare-platform/-/blob/master/k8s/keycloak-values.yaml

```
$ helm repo update

## Install Keycloak Helm Chart
$ helm install keycloak -f keycloak-values.yaml --version 8.0.0 codecentric/keycloak

## To redeploy keycloak:
$ helm upgrade keycloak -f keycloak-values.yaml codecentric/keycloak
```

The output of the above execution will contain information on how to connect to Keycloak from a kubernetes host, or how to expose Keycloak UI through a port for external admin access. Typically Keycloak admin panel is located at "/auth/" url path.

An administrator must open /auth from a browser and create a new realm named "dare". To apply the standard DARE policy, In the new realm register the "dare-login" component as a client, use "confidential" as strategy and save the secret. This is copied in "dare-login-dp.yaml[55]" (also in the appendix) as the value of "CLIENT_SECRET". Next, go back to Keycloak panel, create an admin (username "admin" in our example) and a password and update the values in "dare-login-dp.yaml" for "ADMIN_USERNAME" and "ADMIN_PASSWORD" respectively.

```
$ kubectl create -f dare-login-dp.yaml
$ kubectl create -f dare-login-svc.yaml[56]
$ kubectl expose deployment dare-login --type=NodePort --name=dare-login-public
```

### Deploy all

A set of useful scripts (deploy, redeploy, cleanup) are located in "dare-platform/k8s" downloaded earlier. To get the platform in a functional state, run the deploy script and expose the deployments. This includes core DARE components.

```
$ ./deploy.sh
$ kubectl expose deployment d4p-registry --type=NodePort --name=d4p-registry-public
$ kubectl expose deployment dare-login --type=NodePort --name=dare-login-public
$ kubectl expose deployment exec-api --type=NodePort --name=exec-api-public
$ kubectl expose deployment exec-registry --type=NodePort --name=exec-registry-public
$ kubectl expose deployment playground --type=NodePort --name=playground-public
```

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

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

# 3 Capacity planning and scaling

The type of resources required for the K8s installation are computational (cores, RAM), network (external IPs and inter-VM connectivity) and disk storage. Some of the VMs (at least one, preferably two or more) will take the role of supporting the K8s system itself. The rest will be used as K8s nodes. The amount of resources per VM depends on the requirements and expectations of the process element (PE). If the expected workloads contain computationally intensive PEs, at least some of the node VMs must be provisioned with more CPU, RAM and/or storage.

The amount of resources required for scientific data processing cannot be easily anticipated with sufficient accuracy. It depends on the DARE components being executed as well as the size and structural complexity of the operation (e.g., the depth and width of the workflow graph), the size of the data or the complexity of the algorithms. This was confirmed by executing Seismological and Meteorological workflows on testbed and production and monitoring the systems performance.

Although, DARE can be installed on one host, a minimum of three VMs of moderate power (e.g., 4 cores, 4GB RAM, 30GB storage each) is suggested in order to be able to benefit from the advantages of the platform. Two of the nodes are assigned as K8s masters while the rest act as nodes. In case of increased demand, it is easy to increase the processing and/or storage power of the platform with additional nodes using K8s scaling abilities. After it is launched, operators must be prepared to adjust (usually, extend) resources to demand.

K8s automatically handles physical allocation, as long as there are enough resources to allocate. For example, if a component instance requires unusual amounts of RAM, K8s attempts to execute it on a VM where this amount of RAM is available. The operators have to make sure that these nodes exist and enough of them are available. Monitoring can provide good indications on which resources are insufficient. In this case, operators must be able to support the platform by acquiring additional resources or allocating more of the computing and storage power available on the same cluster.

The requirements of the Platform components are primarily related to the traffic handled by the service (e.g., users served per some time unit) and, consequently, the scope of the service (local or public). The requirements of DARE management components strongly affect the storage capacity planning (size and availability of shared, persistent storage), while the requirements of the DARE execution environments affect the computational power planning (quantity and quality of nodes).

## 3.1 Computational Power

In order to add computational resources, new VMs have to be provisioned. For instance, in case of an IaaS cloud, we can update provisioning scripts by adding definitions of the extra resources and re-run them against the IaaS API, as described in “IaaS Provisioning” section. When this process is completed, K8s must become aware of the new nodes in order to utilize them.

Below we illustrate how to connect a new worker node to K8s. First, token is acquired from the admin host, or by executing the whole join command:

```
# Check if a token is there
$ sudo kubeadm token list
# If the token is expired, issue a new one
$ sudo kubeadm token create
```

```
# Get the join command
$ sudo kubeadm token create --print-join-command
```

Keep the join command for the next step. It must look something like this:

```
kubeadm join 10.0.0.1:6443 \
--token qt57zu.wuvqh64un13trr7x \
--discovery-token-ca-cert-hash
sha256:5ad014cad868fdfe9388d5b33796cf40fc1e8c2b3dccaebff0b066a0532e8723
```

Log on the new node, install docker, kubeadm and kubelet and join the cluster:

```
# Install software
$ sudo apt install -y apt-transport-https
$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
$ sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"
$ sudo apt update && sudo apt install -y kubelet kubeadm docker.io

# Join the cluster (paste the join command here)
$ kubeadm join ...
```

At this point, the new node is part of the K8s cluster.

If the compute resources are underutilized, we can remove some nodes and decommission them to reduce maintenance costs.

On the admin host:

```
# Get a list of the nodes
$ sudo kubeadm get nodes
# Migrate pods from the node and prevent from getting new ones
$ sudo kubectl drain <node-name> --delete-local-data --ignore-daemonsets
$ sudo kubectl cordon <node-name
```

On the node to be removed:

```
# Revert changes made by kubeadm join
$ sudo kubeadm reset
```

## 3.2 Storage

In DARE, K8s relies on Rook to provide a CEPH-RBD[57] layer for storage space shared between pods, which is mandatory for fast execution of workflows and other distributed calculations. With Rook, the process of increasing or reducing storage availability is greatly simplified.

[57] https://docs.ceph.com/en/latest/rbd/

From the perspective of a DARE component implementation, all shared storage is available to all pods all the time, but in reality pods perform well only on data available on the same physical space.Rook CEPH takes care of making data available on pod by allocating PVs (Persistent Volumes) to PVCs (Persistent Storage Claims)[58], but at the end of the day the data has to be copied on the same physical space with the pod.

In order to satisfy requirements for data intensive operations, it may be necessary to extend the storage space on at least some K8s nodes. Physical storage extensions can either be implemented with additional storage volumes on existing VMs, or by provisioning new VMs with larger disk spaces and making them available for K8s.

In case of a new VM with a large physical volume, Rook will take care of the extra storage and will make it available without any additional configurations.

In case of an additional storage device attached to an existing VM, we need to update the file "rook/cluster/examples/kubernetes/ceph/cluster.yaml" in the "rook" directory to look for such devices, by replacing "`useAllDevices: false`" with "`useAllDevices: true`" and applying the change:

```
$ kubectl apply -f cluster.yaml
```

Another option is to explicitly declare the physical storage resources available to CEPH. It is possible to pick specific paths from the filesystem of a node or give away whole block devices, as shown in the example below.

```
  nodes:
   - name: "98.76.54.32"
     directories: # specific directories to use for storage can be specified for each node
     - path: "/rook/storage-dir"
     resources:
       limits:
         cpu: "500m"
         memory: "1024Mi"
       requests:
         cpu: "500m"
         memory: "1024Mi"
   - name: "98.76.54.31"
     devices: # specific devices to use for storage can be specified for each node
     - name: "sdb"
     - name: "sdc"
     config: # configuration can be specified at the node level which overrides the cluster level config
       storeType: filestore
```

[58] https://kubernetes.io/docs/concepts/storage/persistent-volumes/

This approach allows for full control of the physical resources, which would help in supporting calculations with different storage requirements or when the resources are scarce and uneven. The drawback of this method is the high maintenance cost, being incompatible with the auto-discovery mechanism of Rook.

# 4 Authentication and Authorization

The A/A (Authentication/Authorization) requirements of the platform depends on the type of users it supports and the scope of the installation. The aim of an A/A is to protect sensitive data and processes without disturbing accessibility or operations by experts and legitimate information consumers. If the installation is deployed and maintained locally, and is accessible only on premises of an institution, it is preferable to keep a simple A/A scheme, which focuses on providing access to some users. If, on the other hand, the installation is exposed to a wide range of experts and information consumers (e.g. on a multi-institutional or national level), it is required to support multiple access roles focusing also on isolating processes and ensuring that the provided resources will be utilized for the intended purposes.

DARE A/A policy is designed with EOSC integration in mind. Services developed in the scope of European projects on the way to EOSC esp. EOSC-hub[59] , which has proved to be useful, was already integrated in DARE. In specific, the login service is interoperable with EGI-Checkin[60] and for high volume data exchange B2Drop[61] is used. What's more, resources for the testbed and production platforms are provisioned in the scope of the Federated Cloud (EOSC–HUB).

In order to make the service available to EOSC, we set up an SSO (Single Sign-On) environment on Keycloak and then create clients for the appropriate federations. Keycloak offers OAuth2[62] with OpenID Connect[63] (EOSC portal, EGI Check-In, EUDAT B2Access) as well as SAML 2.0[64] (EDUGain) clients. In all these cases, the DARE side will act as a service provider, redirecting potential users to the respective federated login mechanism. 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.

A client (dare-login) is already registered this way and provides access to the DARE login endpoint. It is trivial to start a new OpenID Connect client on the Keycloak admin UI (see "Setting up Platform Components") from the "Client" tab.

[59] https://www.eosc-hub.eu/

[60] https://www.egi.eu/services/check-in/

[61] https://eudat.eu/services/b2drop

[62] https://tools.ietf.org/html/rfc6749

[63] https://openid.net/connect/

[64] http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html

# 5 Monitoring

The platform is monitored by an open-source stack consisting of Prometheus[65], Alertmanager[66] and Grafana[67]. These tools monitor, alert and visualize the performance of the platform, as it was suggested in D5.3. Prometheus acts as the backend for the two other services, triggering Alertmanager and providing the information visualized by Grafana.

DARE installations can use Prometheus to monitor every aspect of the platform. The latest DARE release contains a setup for monitoring the performance of the core DARE components, but more scrape rules can be added in order to monitor other aspects of the execution e.g., statistics on specific data or operations.

Prometheus acts as the information collector and processor, but the service does not inherently distinguish between different types of K8s pods. Statistics and measurements are "scraped"[68] equally from all containers regardless of their type (Platform, DARE management or DARE execution components). Meaningful information is produced by querying Prometheus with context information stored implicitly in pod name prefixes (e.g., a dispel4py pod name may start with "dispel4py_") or other types of systemic metadata. Thus, Grafana can group and visualize performance statistics on all types of pods, regardless of their type.

## 5.1 Setup the stack

Prometheus can be installed on the same K8s cluster or externally. In the first case, setup is easier, but the later has the advantage of being able to monitor large scale failures on the K8s cluster level.

To install prometheus, alertmanager and grafana and expose them for configuration:

```
# Install the stack
$ helm install prometheus stable/prometheus --name dare --namespace monitoring
$ helm install prometheus-operator stable/prometheus-operator \
  --namespace monitoring --values values.yaml[69]
$ helm install grafana stable/grafana --namespace monitoring

# Expose prometheus and grafana
$ kubectl port-forward \
  --namespace default svc/prometheus-kube-prometheus-prometheus 9090:9090
$ export GRAFANA_POD=$(kubectl get pods \
  --namespace default \
  -l "app.kubernetes.io/name=grafana,app.kubernetes.io/instance=grafana" \
  -o jsonpath="{.items[0].metadata.name}")
$ kubectl --namespace default port-forward $GRAFANA_POD 3000:3000

# Get grafana admin password
$ echo "$(kubectl get secret grafana-admin \
  --namespace default -o jsonpath="{.data.GF_SECURITY_ADMIN_PASSWORD}\
  | base64 --decode"
```

[65] https://prometheus.io/

[66] https://prometheus.io/docs/alerting/latest/alertmanager/

[67] https://grafana.com/

[68] The term "scrape" is the one used in Prometheus documentation

[69] https://github.com/helm/charts/blob/master/stable/prometheus/values.yaml

Prometheus is now accessible on port 9090 and Grafana on 3000. To make them talk to each other, log on as "admin" on the Grafana Dashboard, and add Prometheus as a Data source. When prompted for the url, use the internally accessible Prometheus URL, as shown on figure 2, below:
http://prometheus-kube-prometheus-prometheus.default.svc.cluster.local:9090[70]
This will help to secure information, by disallowing access to Prometheus but allowing interested parties to gauge on Grafana-visualized metrics and statistics.

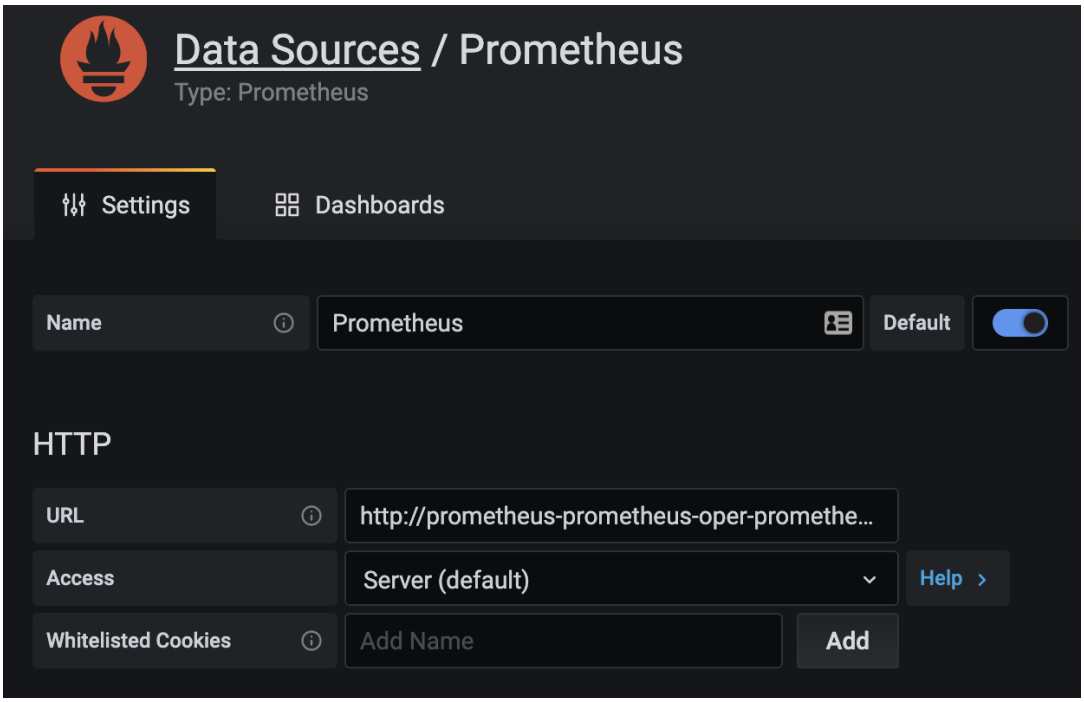


Figure 2: Add prometheus as a Data SOurce on Grafana

Grafana must be further configured by inserting visual elements on its Dashboards. Each visual element is feeded by a data source. The aesthetics and visual aspects of these elements are a matter of preference. The focus of this section is on the content.

When Grafana is configured, expose the deployment:

```
$ kubectl expose deployment grafana --type=LoadBalancer --port=80 \
  --target-port=3000 --protocol=TCP --name=grafana
```

## 5.2 Annotate pods

Any pod with the proper annotations will be automatically observed ("scraped") by Prometheus, thus making the information available to Grafana. Prometheus supports a variety of annotations regarding performance, system information and specific metrics, scraped and processed in a time-series manner. The annotations in "values.yaml" file[71] cover the needs of a DARE installation, but more can be added

[70] Not a URI, but a location, accessible internally. More on the *.cluster.local suffix: https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/

[71] https://github.com/helm/charts/blob/master/stable/prometheus/values.yaml

if needed. A full list of the annotations can be found on the chart's documentation[72].

In order to get prometheus to observe pods, they need to be annotated as shown below:

```
…
metadata:
  annotations:
    prometheus.io/scrape: "true"
    prometheus.io/path: /metrics
    prometheus.io/port:                                                    "9090"
...
```

Each file annotated this way, must be reloaded with "kubectl apply -f".

## 5.3 Alerts

Alertmanager can be configured to provide notifications for various events. It can be configured from an "alertmanager.yaml" file. A simple configuration for an email-based alert policy can be found at the appendix. It is constructed to only report critical events. More elaborate alerts are supported, but they are not necessary in the context of a local installation.

[72] https://github.com/helm/charts/tree/master/stable/prometheus#configuration

# 6 Security and safety

Security requirements are relevant to the value and sensitivity of the data being stored and processed in the platform. A security strategy must consider various factors i.e., protecting infrastructure with firewalls, control access with Keycloak[73] and safeguard HTTPS transports using TLS domain-verified certificates with cert-manager[74]. On top of that, container images deployed on the platform are scanned for vulnerabilities with Harbor. Keycloak and cert-manager configurations are already covered in previous sections.

Regarding the various types of components, security concerts refer to any components exposed directly or indirectly to the outside world. As a security principle, we only expose a service if we must. What's more, each component entering the realm of the platform (i.e., new container images), must be disinfected before put into use. Workflows and applications are executed in isolated environments, while access to provenance, registry or raw data is strictly controlled.

## 6.1 Firewall

It is suggested for all nodes to be protected by firewalls. Different firewalls may fit to different host operations systems. Testbed VMs, for instance, are protected by ufw[75], which is a good fit to Ubuntu 16.04. In any case, we must be careful to set it up so that it does not obstruct K8s functionality.

```
# Install ufw
$ sudo apt install ufw

# Open what is needed for K8s and administration
$ sudo ufw allow ssh
$ sudo ufw allow 179/tcp
$ sudo ufw allow 4789/tcp
$ sudo ufw allow 5473/tcp
$ sudo ufw allow 443/tcp
$ sudo ufw allow 6443/tcp
$ sudo ufw allow 2379/tcp
$ sudo ufw allow 4149/tcp
$ sudo ufw allow 10250/tcp
$ sudo ufw allow 10255/tcp
$ sudo ufw allow 10256/tcp
$ sudo ufw allow 9099/tcp
$ sudo ufw allow 3000/tcp
```

## 6.2 Harbor

Harbor[76] is an open source registry that secures artifacts with policies and role-based access control, ensures images are scanned and free from vulnerabilities, and signs images as trusted. As explained in D5.2[77], when new DARE components or PEs are registered to the platform, they are uploaded to the Harbor registry, scanned and if they turn out safe, marked as available for execution. Harbor is considered an optional platform component, but it is highly suggested for security.

[73] see chapter "Authentication and Authorization" in the present

[74] https://cert-manager.io/docs/

[75] https://help.ubuntu.com/community/UFW

[76] https://goharbor.io/

[77] D5.2 Platform Infrastructure, Usage & Deployment II

We install it through helm:

```
$ sudo helm repo add harbor https://helm.goharbor.io
$ sudo helm fetch harbor/harbor --untar
$ sudo helm install --name dare-harbor .
```

We can modify Harbor by editing the "values.yaml" file, for instance to set up an ingress for the corresponding service, but since we need it for internal checks, it is not mandatory in our context.

## 6.3 Failure prevention and recovery

Failure prevention and recovery is defined as the ability to avoid and, if impossible, recover data and functionality from system failures or other types of disaster. Raw and processed data in DARE, as well as their processing algorithms, are valuable. That's why the primary concern of a failure prevention strategy is to maintain their integrity, but also recover reasonably fast in case of a failure.

Infrastructure failures can be caused by natural or artificial factors and can compromise the efficiency or availability of resources. Cloud operators have developed elaborate techniques in order to manually or automatically recover faulty clusters or VMs by transferring the setup to different hardware. Disk failures are the most critical since they may cause data loss, although cloud vendors utilize replication techniques like RAID and frequent full backups.

K8s features tools with the ability to quickly redeploy a platform. Maintaining automated deployment methods for the full stack is highly suggested for many reasons, one of which is quick recovery. As soon as the VMs are healthy, K8s will auto start. If some operations are still failing, the DARE platform provides a "redeploy" script. Operators and developers must update the deployment scripts whenever the stack is modified.

Data preservation is the first concern, as it will allow the recovery of at least the persistent components of the platform (e.g., provenance, registries, execution API). It also preserves all the materials needed to rebuild the workflows (container images, workflow graphs, input and output of previous executions). For some types of disaster, workflows may have to be rebuilt manually by the cooperators, but if there is no data loss, they are always recoverable.

In order to prevent data loss, it is suggested to install and maintain a backup service like BorgBackup[78], duplicity[79] or rsync, which can backup files like Docker images, K8s persistent volumes, DB dumps and K8s etc[80].

BorgBackup is an open source incremental deduplicated backup tool for full machine backups. It can be installed on each host to periodically sync storage volumes and configuration files to a location topologically unrelated to the cluster.

To set up borg, provision a host with adequate storage space (e.g., a few times larger than the whole cluster space) running a familiar operating system (in our case, ubuntu 16.04).This will be our backup

[78] https://www.borgbackup.org/

[79] http://duplicity.nongnu.org/

[80] https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/

server.

On the backup server (backup.example.org):

```
$ sudo apt install borgbackup
# Assuming /media/backup is a large storage space
$ sudo serve --restrict-to-path /media/backup
```

Make sure the backup server is PPK-accessible and the access key is stored in a known location (e.g., /home/user/.ssh/backup.example.org.key). On each host:

```
$sudo apt install borgbackup

# Context
$ export BORH_PASSPHRASE='some passphrase for the backup not for the key'
$ export BORG_RSH='ssh -i /home/user/.ssh/backup.example.org.key'

# Only the first time, initialize the backup
$ borg init --encryption keyfile username@backup.example.org:dare_full

# To update the backup
$ borg create -p -s -C zlib,6 ::bare-`date +%Y-%m-%d.%H:%M` / \
  -e /tmp -e /root -e /boot

# See all backups
$ borg list

# Recover a file or directory from the backup
$ borg extract ::base-<some date> file/to/restore
```

If some pods are active while being backed up, they will only contribute a snapshot of the process and data to the daily increment. It is suggested to periodically (e.g., weekly) pause all K8s pods and execute a full backup, while keeping the daily backups on schedule. On top of that, it is a good practice to periodically capture DB and registry dumps and store them at a place in the host included in the backup path.

The most important resource to back up is the persistent storage. Normally, the Rook-ceph cluster is not accessible outside K8s, but we can configure it to be mountable from inside the master host as long as "`hostNetwork: true`" (see K8s Network in the Appendix).

```
$ mount -t ceph -o name=admin,secret=$secret $mon_endpoints:/ /ceph_mount
```

# 7 Conclusion

The DARE platform aims to support scientific communities in various ways i.e., assist their effort to tackle arising data processing challenges like workflows, parallelization, timely operations and intensive computations, while providing novel features like provenance. All these are offered in a complete and functional platform which encourages collaboration, reusability of valuable resources (both computational and scientific), consistency and robustness.

The scope of DARE varies depending on the context in which it is required to serve. On one hand, DARE collaborators envisioned large-scale deployments intended to provide computational services to wide academic audiences. On the other, institutions and labs are welcome to utilize the platform to serve their own specific needs. Each of these scopes produces a different class of requirements.

In this document we presented the philosophy of a DARE installation and provided concrete suggestions and guidelines based on the experience of three years from setting up and maintaining DARE testbed and production infrastructures. Since the actual details of these deployments are covered in D5.2, we concentrated on giving advice on a variety of maintenance and deployment subjects. We aspired to enrich advice with concrete examples and highlight tricky details in technical procedures.

To conclude, the DARE platform was an exciting challenge for operators and developers, because it combines a wide range of well established technologies in one stack and was continuously asserted by its potential users, the scientific communities. We primarily adopted tested, mature technologies when building the infrastructure and platform, so that developers and engineers can alleviate their efforts on solving the issues of the higher DARE layers. Our guidelines are based on tried out industry standards only when they fit well to DARE architecture.

Last but not least, as the DARE platform matures, policies and operations are due to adjustments or replacements. New technologies may cause the re-evaluation of some of our choices and new versions of incorporated software may be required in the future. DARE will be challenged to adapt to an environment of evolving technologies and emerging scientific demands. This will be good news, though. It means that scientific communities will embrace it and put it to use.

# 8 Appendix

## K8s Network

```
apiVersion: v1
kind: List
items:
  - apiVersion: v1
    kind: ServiceAccount
    metadata:
      name: weave-net
      annotations:
        cloud.weave.works/launcher-info: |-
          {
            "original-request": {
              "url": "/k8s/net?k8s-version=1.15.3",
              "date": "Fri Dec 18 2020 11:28:52 GMT+0000 (UTC)"
            },
            "email-address": "support@weave.works"
          }
      labels:
        name: weave-net
      namespace: kube-system
  - apiVersion: rbac.authorization.k8s.io/v1
    kind: ClusterRole
    metadata:
      name: weave-net
      annotations:
        cloud.weave.works/launcher-info: |-
          {
            "original-request": {
              "url": "/k8s/net?k8s-version=1.15.3",
              "date": "Fri Dec 18 2020 11:28:52 GMT+0000 (UTC)"
            },
            "email-address": "support@weave.works"
          }
      labels:
        name: weave-net
    rules:
      - apiGroups:
          - ''
        resources:
          - pods
          - namespaces
          - nodes
        verbs:
          - get
          - list
          - watch
      - apiGroups:
          - networking.k8s.io
        resources:
          - networkpolicies
        verbs:
          - get
          - list
          - watch
      - apiGroups:
          - ''
```

```
      resources:
        - nodes/status
      verbs:
        - patch
        - update
- apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRoleBinding
  metadata:
    name: weave-net
    annotations:
      cloud.weave.works/launcher-info: |-
        {
          "original-request": {
            "url": "/k8s/net?k8s-version=1.15.3",
            "date": "Fri Dec 18 2020 11:28:52 GMT+0000 (UTC)"
          },
          "email-address": "support@weave.works"
        }
    labels:
      name: weave-net
  roleRef:
    kind: ClusterRole
    name: weave-net
    apiGroup: rbac.authorization.k8s.io
  subjects:
    - kind: ServiceAccount
      name: weave-net
      namespace: kube-system
- apiVersion: rbac.authorization.k8s.io/v1
  kind: Role
  metadata:
    name: weave-net
    annotations:
      cloud.weave.works/launcher-info: |-
        {
          "original-request": {
            "url": "/k8s/net?k8s-version=1.15.3",
            "date": "Fri Dec 18 2020 11:28:52 GMT+0000 (UTC)"
          },
          "email-address": "support@weave.works"
        }
    labels:
      name: weave-net
    namespace: kube-system
  rules:
    - apiGroups:
        - ''
      resourceNames:
        - weave-net
      resources:
        - configmaps
      verbs:
        - get
        - update
    - apiGroups:
        - ''
      resources:
        - configmaps
      verbs:
        - create
- apiVersion: rbac.authorization.k8s.io/v1
```

```
    kind: RoleBinding
    metadata:
      name: weave-net
      annotations:
        cloud.weave.works/launcher-info: |-
          {
            "original-request": {
              "url": "/k8s/net?k8s-version=1.15.3",
              "date": "Fri Dec 18 2020 11:28:52 GMT+0000 (UTC)"
            },
            "email-address": "support@weave.works"
          }
      labels:
        name: weave-net
      namespace: kube-system
    roleRef:
      kind: Role
      name: weave-net
      apiGroup: rbac.authorization.k8s.io
    subjects:
      - kind: ServiceAccount
        name: weave-net
        namespace: kube-system
  - apiVersion: apps/v1
    kind: DaemonSet
    metadata:
      name: weave-net
      annotations:
        cloud.weave.works/launcher-info: |-
          {
            "original-request": {
              "url": "/k8s/net?k8s-version=1.15.3",
              "date": "Fri Dec 18 2020 11:28:52 GMT+0000 (UTC)"
            },
            "email-address": "support@weave.works"
          }
      labels:
        name: weave-net
      namespace: kube-system
    spec:
      minReadySeconds: 5
      selector:
        matchLabels:
          name: weave-net
      template:
        metadata:
          labels:
            name: weave-net
        spec:
          containers:
            - name: weave
              command:
                - /home/weave/launch.sh
              env:
                - name: HOSTNAME
                  valueFrom:
                    fieldRef:
                      apiVersion: v1
                      fieldPath: spec.nodeName
              image: 'docker.io/weaveworks/weave-kube:2.7.0'
              readinessProbe:
```

```
        httpGet:
          host: 127.0.0.1
          path: /status
          port: 6784
      resources:
        requests:
          cpu: 50m
          memory: 100Mi
      securityContext:
        privileged: true
      volumeMounts:
        - name: weavedb
          mountPath: /weavedb
        - name: cni-bin
          mountPath: /host/opt
        - name: cni-bin2
          mountPath: /host/home
        - name: cni-conf
          mountPath: /host/etc
        - name: dbus
          mountPath: /host/var/lib/dbus
        - name: lib-modules
          mountPath: /lib/modules
        - name: xtables-lock
          mountPath: /run/xtables.lock
    - name: weave-npc
      env:
        - name: HOSTNAME
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: spec.nodeName
      image: 'docker.io/weaveworks/weave-npc:2.7.0'
      resources:
        requests:
          cpu: 50m
          memory: 100Mi
      securityContext:
        privileged: true
      volumeMounts:
        - name: xtables-lock
          mountPath: /run/xtables.lock
  dnsPolicy: ClusterFirstWithHostNet
  hostNetwork: true
  hostPID: true
  priorityClassName: system-node-critical
  restartPolicy: Always
  securityContext:
    seLinuxOptions: {}
  serviceAccountName: weave-net
  tolerations:
    - effect: NoSchedule
      operator: Exists
    - effect: NoExecute
      operator: Exists
  volumes:
    - name: weavedb
      hostPath:
        path: /var/lib/weave
    - name: cni-bin
      hostPath:
```

```
              path: /opt
          - name: cni-bin2
            hostPath:
              path: /home
          - name: cni-conf
            hostPath:
              path: /etc
          - name: dbus
            hostPath:
              path: /var/lib/dbus
          - name: lib-modules
            hostPath:
              path: /lib/modules
          - name: xtables-lock
            hostPath:
              path: /run/xtables.lock
              type: FileOrCreate
    updateStrategy:
      type: RollingUpdate
```

## mpi-operator-0.1.0/deploy/3-mpi-operator.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mpi-operator
  namespace: mpi-operator
  labels:
    app: mpi-operator
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mpi-operator
  template:
    metadata:
      labels:
        app: mpi-operator
    spec:
      serviceAccountName: mpi-operator
      containers:
      - name: mpi-operator
        image: mpioperator/mpi-operator:0.1.0
        args: [
          "-alsologtostderr",
          "--gpus-per-node", "8",
          "--kubectl-delivery-image",
          "mpioperator/kubectl-delivery:0.1.0"
```

## mandatory.yaml

```
apiVersion: v1
kind: Namespace
metadata:
```

```
  name: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx

---

kind: ConfigMap
apiVersion: v1
metadata:
  name: nginx-configuration
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx

---
kind: ConfigMap
apiVersion: v1
metadata:
  name: tcp-services
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx

---
kind: ConfigMap
apiVersion: v1
metadata:
  name: udp-services
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: nginx-ingress-serviceaccount
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx

---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  name: nginx-ingress-clusterrole
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
rules:
  - apiGroups:
      - ""
    resources:
      - configmaps
      - endpoints
      - nodes
```

```
      - pods
      - secrets
    verbs:
      - list
      - watch
  - apiGroups:
      - ""
    resources:
      - nodes
    verbs:
      - get
  - apiGroups:
      - ""
    resources:
      - services
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - ""
    resources:
      - events
    verbs:
      - create
      - patch
  - apiGroups:
      - "extensions"
      - "networking.k8s.io"
    resources:
      - ingresses
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - "extensions"
      - "networking.k8s.io"
    resources:
      - ingresses/status
    verbs:
      - update

---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: Role
metadata:
  name: nginx-ingress-role
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
rules:
  - apiGroups:
      - ""
    resources:
      - configmaps
      - pods
      - secrets
      - namespaces
    verbs:
```

```
      - get
  - apiGroups:
      - ""
    resources:
      - configmaps
    resourceNames:
      # Defaults to "<election-id>-<ingress-class>"
      # Here: "<ingress-controller-leader>-<nginx>"
      # This has to be adapted if you change either parameter
      # when launching the nginx-ingress-controller.
      - "ingress-controller-leader-nginx"
    verbs:
      - get
      - update
  - apiGroups:
      - ""
    resources:
      - configmaps
    verbs:
      - create
  - apiGroups:
      - ""
    resources:
      - endpoints
    verbs:
      - get

---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: RoleBinding
metadata:
  name: nginx-ingress-role-nisa-binding
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: nginx-ingress-role
subjects:
  - kind: ServiceAccount
    name: nginx-ingress-serviceaccount
    namespace: ingress-nginx

---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: nginx-ingress-clusterrole-nisa-binding
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: nginx-ingress-clusterrole
subjects:
  - kind: ServiceAccount
    name: nginx-ingress-serviceaccount
    namespace: ingress-nginx
```

```
---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-ingress-controller
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/name: ingress-nginx
      app.kubernetes.io/part-of: ingress-nginx
  template:
    metadata:
      labels:
        app.kubernetes.io/name: ingress-nginx
        app.kubernetes.io/part-of: ingress-nginx
      annotations:
        prometheus.io/port: "10254"
        prometheus.io/scrape: "true"
    spec:
      # wait up to five minutes for the drain of connections
      terminationGracePeriodSeconds: 300
      serviceAccountName: nginx-ingress-serviceaccount
      nodeSelector:
        kubernetes.io/os: linux
      containers:
        - name: nginx-ingress-controller
          image:               quay.io/kubernetes-ingress-controller/nginx-ingress-
controller:0.30.0
          args:
            - /nginx-ingress-controller
            - --configmap=$(POD_NAMESPACE)/nginx-configuration
            - --tcp-services-configmap=$(POD_NAMESPACE)/tcp-services
            - --udp-services-configmap=$(POD_NAMESPACE)/udp-services
            - --publish-service=$(POD_NAMESPACE)/ingress-nginx
            - --annotations-prefix=nginx.ingress.kubernetes.io
          securityContext:
            allowPrivilegeEscalation: true
            capabilities:
              drop:
                - ALL
              add:
                - NET_BIND_SERVICE
            # www-data -> 101
            runAsUser: 101
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: POD_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
          ports:
```

```
            - name: http
              containerPort: 80
              protocol: TCP
            - name: https
              containerPort: 443
              protocol: TCP
          livenessProbe:
            failureThreshold: 3
            httpGet:
              path: /healthz
              port: 10254
              scheme: HTTP
            initialDelaySeconds: 10
            periodSeconds: 10
            successThreshold: 1
            timeoutSeconds: 10
          readinessProbe:
            failureThreshold: 3
            httpGet:
              path: /healthz
              port: 10254
              scheme: HTTP
            periodSeconds: 10
            successThreshold: 1
            timeoutSeconds: 10
          lifecycle:
            preStop:
              exec:
                command:
                  - /wait-shutdown

---

apiVersion: v1
kind: LimitRange
metadata:
  name: ingress-nginx
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
spec:
  limits:
  - min:
      memory: 90Mi
      cpu: 100m
    type: Container
```

## keycloak-values.yaml

```
    init:
  image:
    repository: busybox
    tag: 1.31
    pullPolicy: IfNotPresent
  resources: {}

keycloak:
  replicas: 1
```

```
  image:
    repository: docker.io/jboss/keycloak
    tag: "10.0.2"
    pullPolicy: IfNotPresent

  hostAliases: []
  enableServiceLinks: false
  podManagementPolicy: Parallel
  restartPolicy: Always

  ## The path keycloak will be served from. To serve keycloak from the root path,
use two quotes (e.g. "").
  basepath: auth

  ## Additional init containers, e. g. for providing custom themes
  extraInitContainers: |

  ## Additional sidecar containers, e. g. for a database proxy, such as Google's
cloudsql-proxy
  extraContainers: |
  extraArgs:     "-Dkeycloak.profile.feature.admin_fine_grained_authz=enabled     -
Dkeycloak.profile.feature.token_exchange=enabled"

  ## Username for the initial Keycloak admin user
  username: keycloak
  password: "AdminPasswordHere"

  ## Allows the specification of additional environment variables for Keycloak
  extraEnv: |
    - name: PROXY_ADDRESS_FORWARDING
      value: "true"

  livenessProbe: |
    httpGet:
      path: {{ if ne .Values.keycloak.basepath "" }}/{{ .Values.keycloak.basepath
}}{{ end }}/
      port: http
    initialDelaySeconds: 300
    timeoutSeconds: 5
  readinessProbe: |
    httpGet:
      path: {{ if ne .Values.keycloak.basepath "" }}/{{ .Values.keycloak.basepath
}}{{ end }}/realms/master
      port: http
    initialDelaySeconds: 30
    timeoutSeconds: 1

  ## Add additional volumes and mounts, e. g. for custom themes
  extraVolumes: |
  extraVolumeMounts: |

  ## Add additional ports, eg. for custom admin console
  extraPorts: |

  service:
    annotations: {}
    labels: {}

    ## ServiceType
    type: ClusterIP
```

```
    httpPort: 80
    httpNodePort: ""
    httpsPort: 8443
    httpsNodePort: ""

  ## Persistence configuration
  persistence:
    # If true, the Postgres chart is deployed
    deployPostgres: true
    dbVendor: postgres
    dbUser: keycloak
    dbPassword: ""

postgresql:
  ### PostgreSQL User to create.
  postgresqlUsername: keycloak

  ## PostgreSQL Password for the new user.
  postgresqlPassword: "ChangeThisPassword"

  ## PostgreSQL Database to create.
  postgresqlDatabase: keycloak

  ## Persistent Volume Storage configuration.
  persistence:
    enabled: true
```

## dare-login-dp.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
        env:
        - name: CLIENT_ID
          value: dare-login
        - name: CLIENT_SECRET
          value: 8f6f4d70-b159-44a7-a520-f8b28bfd4bd8
        - name: ADMIN_USERNAME
          value: admin
```

```
        - name: ADMIN_PASSWORD
          value: admin
        - name: "KEYCLOAK_DOMAIN"
          value: "https://example.org"
```

## alertmanager.yaml

```
global:
  # The smarthost and SMTP sender used for mail notifications.
  smtp_smarthost: 'localhost:25'
  smtp_from: 'alertmanager@example.org'
  smtp_auth_username: 'alertmanager'
  smtp_auth_password: 'password'

# The directory from which notification templates are read.
templates:
- '/etc/alertmanager/template/*.tmpl'

# The root route on which each incoming alert enters.
route:
  # The labels by which incoming alerts are grouped together.
  group_by: ['alertname', 'cluster', 'service']
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 3h

  # A default receiver
  receiver: dare-mails

  # The child route trees.
  routes:
  # This routes performs a regular expression match on alert labels to
  # catch alerts that are related to a list of services.
  - match_re:
      service: ^(dare-login|d4p-registry|dare-shared-volumes|keycloak|sprov|cwl)$
    receiver: dare-mails


# Inhibition rules allow to mute a set of alerts given that another alert is
# firing.
inhibit_rules:
- source_match:
    severity: 'critical'
  target_match:
    severity: 'warning'
  # Apply inhibition if the alertname is the same.
  equal: ['alertname', 'cluster', 'service']

receivers:
- name: 'dare-mails'
  email_configs:
  - to: 'dare+alerts@example.org'
```