H2020-EINFRA-2017
EINFRA-21-2017 - Platform-driven e-infrastructure innovation
DARE [777413] "Delivering Agile Research Excellence on European e-Infrastructures"

D4.2 Big Data Analytics Toolkit II

Project Reference No	777413 — DARE — H2020-EINFRA-2017 / EINFRA-21-2017
Deliverable	D4.2 Big Data Analytics Toolkit II
Work package	WP4: Big Data Processing and Analytics
Tasks involved	T4.2
Type	DEM: Demonstrator, pilot, prototype
Dissemination Level	PU = Public
Due Date	31/12/2020
Submission Date	30/12/2020
Status	Draft
Editor(s)	Sissy Themeli (NCSR-D), Iraklis Klampanos (NCSR-D)
Contributor(s)	
Reviewer(s)	Fraunhofer
Document description	The report accompanies the software deliverable for the Big Data Analytics components integrated in the DARE platform. It provides a short description on the different components, their role within the DARE platform and their current status of maturity. It also provides links to relevant code and documentation.

Document Revision History

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
1	09/09/2020	Initial version	S. Themeli (NCSR-D)
2	16/12/2020	Updates and edits throughout	I. Klampanos (NCSR-D)
3	30/12/2020	Final	I. Klampanos (NCSR-D)

Executive Summary

This report acts as an overview of the technical developments pertaining to the Big Data Analytics toolkit incorporated in the DARE platform.
The document summarises the functionality of the relevant components, provides information on their deployment and availability and refers to the relevant code and documentation available through the project's GitLab repositories.

Table of Contents

1. 4

1.1 Purpose and scope **5**

1.2 Relationship with other Work Packages and Deliverables **5**

1.3 Methodology and Structure of the Deliverable **5**

2. 4

2.1 Integrated DARE platform **6**

2.2 DARE platform deployment **7**

i. 6

ii. 6

iii. 10

3. 11

1. Introduction

The present report acts as an overview of the technical developments pertaining to the Big Data Analytics toolkit incorporated in the DARE platform. It summarises the functionality of the relevant components, provides information on their deployment and availability and refers to the relevant code and documentation available through the project's GitLab repositories.

○ 1.1 Purpose and scope

This deliverable reports on the basis of the DARE platform, which is a collection of dockerised components deployed and managed by Kubernetes.

○ 1.2 Relationship with other Work Packages and Deliverables

Other components and integration activities effectively take place on the technical basis provided by the big-data and analytics foundation. As the DARE platform has undergone a number of design and implementation steps during the project, this deliverable has some overlap with D4.8 – Integrated software stack and the semantic registry II.

○ 1.3 Methodology and Structure of the Deliverable

We provide a list of DARE-specific and 3rd-party components. Due to the relationship of the components and the Kubernetes underlying system to deployment, we also provide basic deployment and installation instructions.

2. DARE Big Data Analytics Components

The components comprising the DARE Big Data Analytics Toolkit, as well as the methodology followed have initially been based on the Big Data Integrator (BDI) platform, which incorporated initial versions of the tools required for serving the DARE use cases. These include both general-purpose Big Data Processing tools and domain-specific tools addressing the needs of the participating communities. The tools integrated at this point are described in the following subsection. The Big Data Analytics Toolkit is available in the dare-platform GitLab Repository[1].

[1] https://gitlab.com/project-dare/dare-platform

2.1 Integrated DARE platform

The DARE platform comprises tools and systems that enable the easy creation of scalable Big Data applications. It is heavily dependent and built on Docker[2] and Kubernetes[3] technologies, to ensure the ease of packaging, deployment, maintainability and modularity. Moreover, it provides a wide variety of systems packaged as Docker images and verified in various use-case scenarios. Therefore, the DARE platform consists of various general-purpose components as well as the DARE components. The DARE platform is extensively documented online, at https://project-dare.gitlab.io/dare-platform/, which includes installation/ deployment documentation, use examples, etc.

The general-purpose components provide generic functionality to the platform such as storage, monitoring and authentication while the DARE-specific components implement elements of the DARE Architecture (detailed in D2.2). All the components are containerised and by design share common resources of the cluster.

The core DARE components, currently deployed in the operational environment are:

- Dispely4py Information Registry[4], which is a Django RESTful Web Service that allows the registration, sharing and manipulation of dispel4py workflows. After registration, workflows can be accessed and executed by name.
- CWL Workflow Registry[5], which is a Django RESTful Web Service, enabling the use of different Docker execution environments and the registration of CWL workflows. The environments and workflows can be accessed and executed by name and version.
- DARE Login API[6], a central Authentication component that uses Keycloak[7] in the backend.
- DARE Execution API[8], which is a Flask-based[9] RESTful Web Service used to instantiate dispel4py and CWL execution environments, folders and files handing/listing, uploading and downloading files, etc.
- S-prov[10], which is a RESTful Web Service allowing the storage and retrieval of Provenance logs.
- Playground[11], which is a RESTful Web Service providing a testing environment to the users.
- Semantic-data-discovery[12], which is a RESTful Web Service for searching registered data in the DARE data catalogue.
- sprov-viewer[13], which is a graphical user interface (GUI) to facilitate the search and visualisation of provenance logs.

These components make use of 3rd-party underlying database systems, such as:

- MySQL databases, used by Dispel4py and CWL Registries
- MongoDB, used by the Provenance API

[2] https://www.docker.com
[3] https://kubernetes.io
[4] https://gitlab.com/project-dare/d4p-registry
[5] https://gitlab.com/project-dare/workflow-registry
[6] https://gitlab.com/project-dare/dare-login
[7] https://www.keycloak.org
[8] https://gitlab.com/project-dare/exec-api
[9] https://flask.palletsprojects.com/en/1.1.x/
[10] https://gitlab.com/project-dare/s-ProvFlow
[11] https://gitlab.com/project-dare/playground
[12] https://gitlab.com/project-dare/semantic-data-discovery
[13] https://gitlab.com/project-dare/s-ProvFlow

- Virtuoso, used as backend to the semantic-data component
- Rook ceph storage, used by the Execution API to provide the users with a Shared File System that can be accessed from the RESTful API as well as during the workflow execution

The Kubernetes components deployed to support the DARE platform are the following:

- Rook ceph, used as Shared File System
- Keycloak, which acts as the Authentication broker, allowing to use 3rd party authentication providers, such as, for instance Google, EGI Check-in[14], and others.
- MPI-operator, used to enable the execution of MPI jobs on the Kubernetes cluster
- Nginx web servers used by multiple DARE components
- Cert-Manager, which manages the issue of SSL certificates.

2.2 DARE platform deployment

i. Operating System specification and requirements

The DARE platform deployment has been tested with the operating system Ubuntu Server 18.04. Version 18.04 version is a long-term support version (LTS), and it is supported until 2028.

We assume that the deployment takes place in a cluster of machines and each machine has two block devices. The first one (/dev/vda) will be used for the installation of the operating system while the second one (/dev/vdb) will be used by Ceph and will remain unformatted. Every machine in the cluster is identical. Moreover, at least one of the machines should have two network interfaces (say, eth0 and eth1). The first one will be used for the communication with the other cluster nodes while the second one will be used for installing the bare-metal load balancer.

The workflow for installing the basic operating system is the usual one. The operator has to provide the username and password for the initial user when asked by the Ubuntu installer. Moreover, it has to provide the network details such as hostname, domain name, IP, netmask, gateway and DNS. These details can be provided by the network operator of the data centre.
After the successful installation of the operating system the passwordless ssh root access must be activated.

The aforementioned step can be skipped if the machines are virtual and the operating system is created from a template image.

ii. Kubernetes and general-purpose components installation

As already mentioned, the DARE platform is based on Kubernetes and Docker therefore, before installing the DARE platform some basic components should be available in the infrastructure. Below, the necessary steps are provided:

1. Kubernetes setup
    a. Install docker: *sudo apt install docker.io*
    b. Enable docker: *sudo systemctl enable docker*
    c. Add Kubernetes signing key:

    curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add

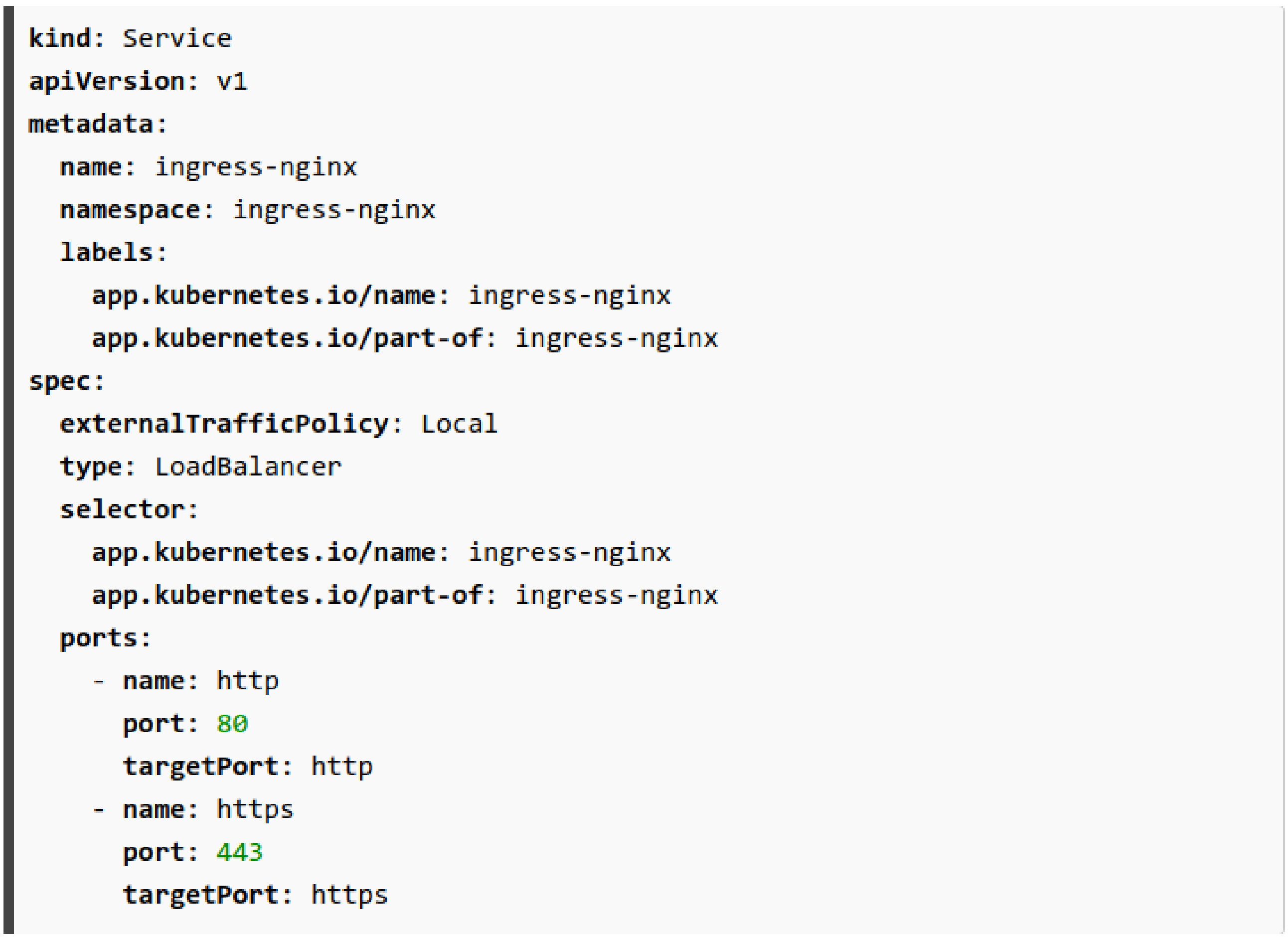
[14] https://www.egi.eu/services/check-in/

   d. Add Xenial Kubernetes Repository: *sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"*
   e. Install Kubeadm : *sudo apt install kubeadm=1.15.3-00 kubectl=1.15.3-00 kubelet=1.15.3-00*
   f. Initialize Kubernetes on the master node: *sudo kubeadm init*
   g. Start using your cluster : *mkdir -p $HOME/.kube & sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config & sudo chown $(id -u):$(id -g) $HOME/.kube/config*
   h. Deploy a Pod Network through the master node : *kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"*
   i. Run: *kubectl taint nodes --all node-role.kubernetes.io/master-*

2. Install the MPI-Operator (v 0.1.0)
   a. wget https://github.com/kubeflow/mpi-operator/archive/0.1.0.tar.gz
   b. tar -xvzf 0.1.0.tar.gz
   c. cd mpi-operator-0.1.0/deploy/
   d. In 3-mpi-operator.yaml, change the two images (mpi-operator & kubectl-delivery) from latest to 0.1.0
   e. Deploy mpi-operator:
      i. kubectl create -f 0-crd.yaml
      ii. kubectl create -f 1-namespace.yaml
      iii. kubectl create -f 2-rbac.yaml
      iv. kubectl create -f 3-mpi-operator.yaml

3. Deploy the Rook Shared File System (release-0.8)
   a. git clone https://github.com/rook/rook.git
   b. git checkout release-0.8
   c. cd rook/cluster/examples/kubernetes/ceph
      i. kubectl create -f operator.yaml
      ii. kubectl create -f cluster.yaml
      iii. kubectl create -f filesystem.yaml
      iv. kubectl create -f storageclass.yaml
   d. NOTE: rook-ceph-block storageclass is not default. To set as default: *kubectl patch storageclass rook-ceph-block -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'*

4. Deploy Ingress
   a. kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/130af33510882ae62c89277f2ad0baca50e0fafe/deploy/static/mandatory.yaml
   b. Check ingress controller: kubectl get pods -n ingress-nginx
   c. mkdir ingress-deployment && cd ingress-deployment
   d. vim nginx-ingress.yaml
   e. Add the following in the file:



```yaml
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
```

Figure 1: Ingress yaml specification

f. kubectl apply -f nginx-ingress.yaml
g. Check the created service: kubectl get svc -n ingress-nginx

5. Install Helm & Tiller

DARE uses the Helm package manager (https://helm.sh/) for Kubernetes to install and manage some of the external packages it uses. This facilitates the installation and upgrade of external components and prevents duplicated work on Kubernetes descriptors for well-known applications. To use Helm, we need to install the helm command and corresponding service (Tiller) first. Summary:

a. Download 3.1.1 release from https://github.com/helm/helm/releases
b. Install helm binary to /usr/local/bin
c. Initialize helm and Tiller
d. Update package sources

```
## Download & unpack 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

## move to path

$ 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
```

6. Install cert-manager

To maintain certificates for the externally reachable services and pages, DARE uses the Let's Encrypt Certification Authority (https://letsencrypt.org/) through its ACME protocl (https://tools.ietf.org/html/rfc8555). The Cert-Manager addon (https://cert-manager.io) automates this process even further so that certificates are automatically issued, configured in the ingress and updated based on annotations in our Kubernetes descriptors. For installation, we use the official Helm package from Helm hub (https://hub.helm.sh/charts/jetstack/cert-manager). Summary:

a. Install the custom resource definitions required for cert-manager
b. Activate the Jetstack Helm repository
c. Install the Cert-Manager package
d. Add a clusterissue for letsencrypt

```
## install custom resource definitions

$ kubectl apply --validate=false -f https://raw.githubusercontent.com/jetstack/cert-manager/release-0.14/deploy/manifests/00-crds.yaml

## 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 https://raw.githubusercontent.com/kubernetes/k8s.io/master/cert-manager/letsencrypt-prod.yaml
```

7. Install Keycloak (WIP)

```
## Add codecentric package source and update package sources.

$ helm repo add codecentric https://codecentric.github.io/helm-charts

$ helm repo update
```

## Install Keycloak Helm Chart

$ helm install keycloak -f keycloak-values.yaml --version 8.0.0 codecentric/keycloak

iii. DARE platform installation

After having successfully executed the previous steps, the infrastructure should be ready in order to install the dare-platform. The following instructions are related to the DARE components deployment.

a. git clone https://gitlab.com/project-dare/dare-platform.git
b. Check out the latest release
c. cd dare-platform/k8s/ or cd dare-platform/k8s-operational
    i. The yaml files in k8s directory are associated to the testbed while the k8s-operational to the operational environments.
    ii. In order to install the DARE platform in a different environment, the yaml files should be adjusted accordingly
d. execute the script deploy.sh
e. Expose deployments:
    i. kubectl expose deployment d4p-registry --type=NodePort --name=d4p-registry-public
    ii. kubectl expose deployment dare-login --type=NodePort --name=dare-login-public
    iii. kubectl expose deployment exec-api --type=NodePort --name=exec-api-public
    iv. kubectl expose deployment exec-registry --type=NodePort --name=exec-registry-public
    v. kubectl expose deployment playground --type=NodePort --name=playground-public
    vi. kubectl expose deployment semantic-data --type=NodePort --name=semantic-data-public
    vii. kubectl expose deployment workflow-registry --type=NodePort --name=workflow-registry-public
f. If volumes are not mounted –> systemctl restart kubelet

The DARE platform supports the collaboration of the users and provides a JupyterHub component. In order to deploy it, perform the following steps:

a. Create secret token for jupyterhub-config.yaml: openssl rand -hex 32
b. Copy the token and paste it in the aforementioned yaml file in the secretToken field
c. Run the following
    i. helm repo add jupyterhub https://jupyterhub.github.io/helm-chart/
    ii. helm repo update
    iii. kubectl create namespace jupyterhub
    iv. helm upgrade --install jupyterhub jupyterhub/jupyterhub --namespace jupyterhub --version=0.9.0 --values jupyterhub-config.yaml

The latest version of the above installation instructions of the Kubernetes and DARE components are always available online, in the DARE platform's microsite[15].

3. Summary

The DARE platform is a deployment cloud-native platform designed with the objective to be easily installable and configurable automatically using Kubernetes. The components are updated to their final stable versions. Keycloak service is integrated to the DARE platform for user authentication. It is extended to also use 3rd party identity providers, thus enabling the integration with EGI Checkin and other providers.

[15] https://project-dare.gitlab.io/dare-platform/installation/