H2020-EINFRA-2017

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

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

Big Data Analytics Toolkit I

Project Reference No	777413 — DARE — H2020-EINFRA-2017 / EINFRA-21-2017
Deliverable	D4.1 Big Data Analytics Toolkit I
Work package	WP4: Big Data Processing and Analytics
Tasks involved	T4.1
Type	DEM: Demonstrator, pilot, prototype
Dissemination Level	PU = Public
Due Date	30/06/2019 Deadline extended to 31/07/2019 in agreement with PO
Submission Date	31/07/2019
Status	Final Draft
Editor(s)	Antonis Koukourikos (NCSR-D)
Contributor(s)	Angelos Charalambidis (NCSR-D), Giannis Mouchakis (NCSR-D), André Germund (FRAUNHOFER)
Reviewer(s)	Wim Som de Cerff (KNMI)
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
0.1	24/06/2019	ToC	NCSR-D
0.5	15/07/2019	Section 2	NCSR-D, FRAUNHOFER
0.8	22/07/2019	Draft for Internal Review	NCSR-D
0.9	26/07/2019	Internal Review	KNMI
1.0	29/07/2019	Review comments addressed and document finalised	NCSR-D

Executive Summary

The present report acts as an overview of the technical developments pertaining to the Big Data Analytics toolkit incorporated in the DARE platform.
The document summarizes 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 Introduction

The document is an accompanying report to the deployment of the first version of the Big Data Analytics toolkit that is incorporated in the DARE Software Stack.

1.1 Purpose and Scope

The purpose of the report is to summarize the progress on T4.1, i.e. the task responsible for developing, customizing and making available the Big Data Analytics components that will be used via the integrated DARE platform.
It provides links to relevant code repositories and documentation and offers guidelines for installing and deploying the toolkit.

1.2 Approach and relationship with other Work Packages and Deliverables

D4.1, a direct deliverable for T4.1, Big Data Analytics Toolkit, encapsulates all activities of the task, designed after communication with several other tasks and work packages.
Specifically, it follows the blueprint provided by WP2 and particularly D2.1, DARE Architecture and Technical Positioning. Additionally, it takes into account the requirements and user stories defined by WP3 and aims to serve the pilots of the two DARE user communities (WP6 and WP7). Finally, the task worked closely with WP5, in order to coordinate the installation and deployment processes.

1.3 Methodology and Structure of the Deliverable

The report accompanies a software deliverable comprising the Big Data Analytics components thus far incorporated in the DARE platform, under the guidelines and design set by the DARE architectural principles (cf. deliverable D2.1 of the project).
Section 2.1 provides a high-level overview of the relevant components and their organization under the Big Data Integrator platform. Section 2.2 summarizes the adopted approach for their deployment, along with the relevant instructions and guidelines.
Finally, Section 3 reports on future steps towards the next platform release and the second version of the Big Data Analytics toolkit itself.

2 DARE Big Data Analytics Components

The components comprising the DARE Big Data Analytics Toolkit are organized as an instantiation of the Big Data Integrator (BDI) platform, which incorporates 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.

2.1 DARE Big Data Integrator Platform

The *Big Data Integrator* (BDI) platform comprises a set of tools and systems that enabled the easy creation of scalable Big Data infrastructures. BDI uses at its core *Docker*[1] technologies to ensure the ease of packaging and deployment. Moreover, it provides a wide variety of systems packaged as Docker images and verified in various use-case scenarios.

DARE BDI platform builds on these proven technologies to provide the core infrastructure of every DARE deployment. In contrast to the BDI platform, DARE platform is based on *Kubernetes*[2] rather than the simpler *Docker Swarm*[3]. Kubernetes provides a more sophisticated deployment strategies and currently is the de-facto resource management framework for deployment containerized scalable applications in the cloud.

The DARE BDI platform is comprised of the general-purpose components and the DARE components.
The general-purpose components provide generic functionality to the platform such as storage, monitoring and authentication while the DARE components realize the DARE architecture. All the components are containerized and share common resources of the cluster.

Currently, the main DARE components that are available and deployable in the DARE platform are the following:

- *dispel4py* that acts as the current implementor of the WaaS (Workflow as a Service) and it used to enact user-defined workflows.
- *S-PROV* that is responsible for collecting, preserving and reporting on provenance information from the workflow execution over the platform.
- *SemaGrow* that is the realization of the federating layer of the various metadata of the platform.
- *d4p-registry* that acts as the registry of the available processing elements of dispel4py
- *data-catalogue* that stores metadata of the datasets available

Apart from the main DARE components there are also deployed supporting systems that facilitate the execution of the main components. These systems are the following:

- *mysql* acting as persistent storage of the d4p-registry
- *mongodb* acting as persistent storage of sprov
- *virtuoso* acting as persistent storage of the data-catalogue
- *sprov-viewer* used as the frontend user interface for visualized provenance from sprov.

The infrastructural components of the DARE platform are the following:

- *ceph* providing the main distributed storage facility for the cluster

[1] https://www.docker.com/
[2] https://kubernetes.io/
[3] https://docs.docker.com/engine/swarm/

- *nginx* acting as the main entry to the platform, namely it redirects requests to the external DARE API to internal services.
- *keycloak* acting as the authentication broker
- *prometheus* and *grafana* for collecting and visualizing metrics of the cluster's performance.
- *mpi-operator* that is used to enable the execution of MPI jobs in the Kubernetes cluster.
- *cert-manager* to manage the issue of SSL certifications.

The DARE BDI platform is publicly available in https://www.gitlab.com/project-dare/dare-bdi

2.2 Deployment and availability

2.2.1 Deployment in Bare-Metal machines

Installation of the operating system

The platform is currently tested with the operating system Ubuntu Server 18.04.[4] The version 18.04 is long term support (LTS) and is supported and receive updates 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 typical 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 center.
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.

Installation of the basic platform

The basic platform includes Docker and Kubernetes. For the installation we use Kubespray 2.9, a set of Ansible[5] playbooks, inventory[6], provisioning tools and domain knowledge for generic OS/Kubernetes cluster configuration management tasks. This procedure ensures that the OS configuration is automated, and the installation is always the same. Prior of executing Kubespray we need to satisfy its dependencies.[7]
First, we need to download the playbooks of Kubespray

```
git clone -b release-2.9 https://github.com/kubernetes-sigs/kubespray.git
cd kubespray
cp -rfp inventory/sample inventory/mycluster
```

Second we apply the following changes:

[4] http://releases.ubuntu.com/18.04
[5] http://docs.ansible.com
[6] https://github.com/kubernetes-incubator/kubespray/blob/master/docs/ansible.md
[7] https://github.com/kubernetes-sigs/kubespray/tree/release-2.9#requirements

(a) In the file `inventory/mycluster/group_vars/all/all.yaml` we set

```
kubelet_load_modules: true
```

and

```
upstream_dns_servers:
  - 8.8.8.8
  - 8.8.4.4
```

(b) In the file `inventory/mycluster/group_vars/k8s-cluster/addons.yml` we set

```
helm_enabled: true
```

The aforementioned settings instruct the Kubespray to also install Helm, a Kubernetes package manager.

(c) In the file `inventory/mycluster/inventory.ini` we set the IPs of every machine in the cluster.

The following commands in the terminal should initialize the installation.

```
sudo pip install -r requirements.txt
ansible-playbook -i inventory/mycluster/inventory.ini --become --become-
user=root -u root cluster.yml
```

After the successful return of the commands one can verify the installation by executing in a master node the following

```
kubectl get pods --all-namespaces
```

The expected answer is that all the pods have status “Running”.

Installation of the Ceph storage system

We also want to install a storage facility that will provide to the Kubernetes cluster persistent, distributed and highly-available block devices. The following instruction will lead to a working Ceph[8] system running in Kubernetes itself and thus in the same nodes of the cluster. Recall that the machines assume a second block device (/dev/vdb) that Ceph will manage. For the installation of Ceph in the Kubernetes environment we use Rook[9].

First we download the sources of Rook in a Kubernets master.

```
git clone -b release-0.9 https://github.com/rook/rook.git
```

[8] https://ceph.com/

[9] https://rook.io/

Then we specify some details in the configuration. In the file `cluster/examples/kubernetes/ceph/operator.yaml` we set:

```
- name: FLEXVOLUME_DIR_PATH
  value: "/var/lib/kubelet/volume-plugins"
```

Lastly we deploy the Rook operator using the following command.

```
kubectl apply -f cluster/examples/kubernetes/ceph/operator.yaml
```

The expected behavior is to successfully create the rook-ceph-operator, rook-ceph-agent and rook-discover pods. Their status should be “Running” and can be checked using

```
kubectl -n rook-ceph-system get pod
```

To deploy the Ceph cluster first in the file `cluster/examples/kubernetes/ceph/cluster.yaml` we set:

```
storage:
  useAllNodes: true
  useAllDevices: false
  deviceFilter: "vdb"
```

We apply our changes using

```
kubectl apply -f cluster/examples/kubernetes/ceph/cluster.yaml
```

Upon successful application the ceph cluster will be deployed and the following command

```
kubectl -n rook-ceph get pod
```

will answer that the rook-ceph-mgr, rook-ceph-mon and rook-ceph-osd pods have all status “Running”.

Finally, we need to activate the ceph block storage as a storageclass in Kubernetes. This can be achieved using the following command.

```
kubectl apply -f cluster/examples/kubernetes/ceph/storageclass.yaml
```

We can set the newly created storageclass as default (which is recommended) using the following

```
kubectl patch storageclass rook-ceph-block -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

2.2.2 Deployment in Kubernetes-ready environments

The following instructions assume the existence of a functional Kubernetes cluster and a default storage class that can be used for acquiring persistent volumes. Most of the managed kubernetes services will provide these dependencies out of the box. On any other case one should consider the previous sections for installing Docker, Kubernetes and Ceph in bare-metal machines.

The platform uses Helm to ease the installation and configuration of the individual components.

First, we should download the sources of the DARE BDI platform using

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

Then, we need to package the platform using

```
helm dep up
mkdir -p /build
helm package . -d /build
```

and install the platform to the Kubernetes cluster using

```
helm install -n dare -f values.yaml /build/dare-bdi-*.tar.gz
```

In order to retrieve the status of the deployment one should use

```
helm status dare
```

The file `values.yaml` contain various configuration options. If not sure one can use the default values provided by the platform (i.e. do not provide any overrides of configuration in the helm install command).

A testbed platform is online and accessible in https://testbed.project-dare.eu

3 Summary and Next Steps

The DARE BDI platform is a deployment cloud-native platform designed with the objective to be easily deployable in a variety of infrastructures. As next steps we plan to package the remaining DARE components as containers and incorporate them into the final deployable platform. We also plan to make the installation and configuration as automated as possible. Moreover, we will maintain the infrastructural components and upgrade them to their latest stable versions. Finally, we will setup cluster authentication and authorization for the platform users.