Deliverable 5.3
Generic Big Data Integrator Instance III

**Dissemination Level** | Public
---|---
**Due Date of Deliverable** | M32
**Actual Submission Date** | M36+, 31/01/2018
**Work Package** | WP5, Big Data Integrator Instances
**Task** | T5.1
**Type** | Other
**Approval Status** | Approved
**Version** | 1.0
**Number of Pages** | 50
**Filename** | D5.3_Generic Big Data Integrator Instance III

**Abstract:** Documentation of the Generic Big Data Integrator Instance. This instance is used for generic functionality and usability tests, while it also targets the Big Data community in general.

The information in this document reflects only the author’s views and the European Community is not liable for any use that may be made of the information contained therein. The information in this document is provided "as is" without guarantee or warranty of any kind, express or implied, including but not limited to the fitness of the information for a particular purpose. The user thereof uses the information at his/ her sole risk and liability.

Project funded by the European Union’s Horizon 2020 Research and Innovation Programme (2014 – 2020)
History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Reason</th>
<th>Revised by</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>20/10/2017</td>
<td>Document structure and initial draft, assuming D5.5 as a starting point.</td>
<td>S. Konstantopoulos</td>
</tr>
<tr>
<td>0.1</td>
<td>20/10/2017</td>
<td>Upgrade platform to the latest version of Docker Engine and Docker Swarm</td>
<td>I. Mouchakis</td>
</tr>
<tr>
<td>0.2</td>
<td>02/11/2017</td>
<td>Improvement of the platform installation scripts.</td>
<td>I. Mouchakis</td>
</tr>
<tr>
<td>0.3</td>
<td>29/11/2017</td>
<td>Added rationale subsections explaining decisions regarding the platform and the network topology.</td>
<td>I. Mouchakis, S. Konstantopoulos</td>
</tr>
<tr>
<td>0.4</td>
<td>10/12/2017</td>
<td>Updated Docker Compose snippets for all BDE components (Section 3)</td>
<td>I. Mouchakis, A Charalambidis, I Ermilov</td>
</tr>
<tr>
<td>0.5</td>
<td>15/12/2017</td>
<td>Testing of the deployment instructions (Section 5) and editorial changes and improvements throughout.</td>
<td>I. Mouchakis, G. Stavrinos, A. Charalambidis, S. Konstantopoulos</td>
</tr>
<tr>
<td>0.6</td>
<td>16/1/2018</td>
<td>Peer review</td>
<td>Hajira Jabeen</td>
</tr>
<tr>
<td>0.7</td>
<td>30/1/2018</td>
<td>Address peer review comments (Section 4).</td>
<td>Jonathan Langens</td>
</tr>
<tr>
<td>1.0</td>
<td>31/1/2018</td>
<td>Final version</td>
<td>I. Mouchakis, S. Konstantopoulos, A. Garatzogianni</td>
</tr>
</tbody>
</table>

Author List

<table>
<thead>
<tr>
<th>Organisation</th>
<th>Name</th>
<th>Contact Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCSR-D</td>
<td>Ioannis Mouchakis</td>
<td><a href="mailto:gmouchakis@iit.demokritos.gr">gmouchakis@iit.demokritos.gr</a></td>
</tr>
<tr>
<td>NCSR-D</td>
<td>Georgios Stavrinos</td>
<td><a href="mailto:gstavrinos@iit.demokritos.gr">gstavrinos@iit.demokritos.gr</a></td>
</tr>
</tbody>
</table>
Executive Summary

This report documents the deployment of the Big Data Integrator Platform and provides instructions on how to reproduce identical deployments. Various configurations and component mixtures of this generic platform will be used in the BDE pilots to serve exemplary use cases of the Horizon 2020 Societal Challenges.

The instructions in this document include (a) installation of the base system; (b) network topology and configuration; and (c) the components available as docker images and how they can be spawned and accessed to create pilot applications.
## Abbreviations and Acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDI</td>
<td>Big Data Integrator</td>
</tr>
<tr>
<td>LOD</td>
<td>Linked Open Data</td>
</tr>
<tr>
<td>RAID</td>
<td>Redundant Array of Independent Disks</td>
</tr>
<tr>
<td>QEMU</td>
<td>Machine Emulator and Virtualizer</td>
</tr>
</tbody>
</table>
# Table of Contents

1. Introduction .......................................................................................................................... 9
   1.1 Purpose .............................................................................................................................. 9
   1.2 Methodology .................................................................................................................... 9

2. Base System .......................................................................................................................... 10
   2.1 Base Installation and Configuration ............................................................................... 11
   2.2 Network Topology .......................................................................................................... 11
   2.3 Virtual Node Installation ............................................................................................... 12
      2.3.1 Ansible ...................................................................................................................... 13
      2.3.2 Docker tooling ........................................................................................................... 14
      2.3.3 Gateway ..................................................................................................................... 15
      2.3.4 Master node ............................................................................................................... 17
      2.3.5 Slave nodes ............................................................................................................... 17
      2.3.6 Master and slave node configuration ......................................................................... 18
   2.4 Recovery from Failures ................................................................................................. 19
      2.4.1 Master node failure ................................................................................................... 19
      2.4.2 Slave node failure .................................................................................................... 19
   2.5 Adding a New Node to a Live Swarm ........................................................................... 20
   2.6 Cloning a Node .............................................................................................................. 22
   2.7 Pilot Swap ....................................................................................................................... 23
   2.8 BDE Cluster SSH Access .............................................................................................. 24
   2.9 Port Forwarding .............................................................................................................. 26

3. Other BDE components ........................................................................................................ 27
   3.1 Apache Spark .................................................................................................................. 28
   3.2 OpenLink Virtuoso .......................................................................................................... 29
   3.3 Semagrow and sevod-scraper ....................................................................................... 30
3.4 4store ................................................................. 31
3.5 GeoTriples .......................................................... 32
3.6 Sextant ............................................................... 33
3.7 Silk ................................................................. 34
3.8 PostGIS ............................................................. 35
3.9 Strabon .............................................................. 36
3.10 Cassandra ......................................................... 36
3.11 Hadoop ............................................................ 38
3.12 Flink ............................................................... 38

4. Installing and Executing BDI Stacks .................................. 43
  4.1 Installing and Launching the BDI IDE ................................. 43
  4.2 Installing a Stack on the BDI ........................................ 44
  4.3 Launching the stack .................................................. 46
  4.4 Monitoring a stack ................................................... 46

5. Conclusion ...................................................................... 48
List of Tables

1. Introduction
   Purpose
   Methodology

2. Base System
   Rationale
   2.1 Base Installation and Configuration
      Rationale
   2.2 Network Topology
      Rationale
   2.3 Virtual Node Installation
      2.3.1 Ansible
      2.3.2 Docker tooling
         Rationale
      2.3.2 Gateway
   2.3.3 Master node
   2.3.4 Slave nodes
   2.3.5 Master and slave node configuration
   2.4 Recovery from Failures
      2.4.1 Master node failure
      2.4.2 Slave node failure
   2.5 Adding a New Node to a Live Swarm
   2.6 Cloning a Node
2.7 Pilot Swap
2.8 BDE Cluster SSH Access
2.9 Port Forwarding

3. Other BDE components
3.1 Apache Spark
3.2 OpenLink Virtuoso
3.3 Semagrow and sevod-scraper
3.4 4store
3.5 GeoTriples
3.6 Sextant
3.7 Silk
3.8 PostGIS
3.9 Strabon
3.10 Cassandra
3.11 Hadoop
3.12 Flink

4. Conclusion
1. Introduction

1.1 Purpose

This report documents the deployment of the generic Big Data Integrator Platform (BDI) at the NCSR-D cluster. Although the examples are based on this particular cluster, the instructions are generic and explanations are provided on how to generalize to different clusters.

1.2 Methodology

Section 4 of this document was first prepared by documenting all steps while the cluster was setup and the BDI platform was deployed. The draft produced in this manner was then given to a third person who was not involved in the first setup; this person executed the whole installation procedure from scratch while we were noting and improving places in the text that proved to be unclear for the third person. The trial was repeated with two further trial subjects who are NCSR-D staff, not working for the BDE action and who had never deployed or used Big Data tools. In the second step of this process the document was updated for the newer versions of the tools used and the process was repeated with one subject from the first phase, one new subject working for BDE and a novice Linux user. All provided comments that helped improve this document.

Section 3 archives the current state of the README and Wiki instructions from the BDE Github repositories.
2. Base System

Our example setup has one gateway node, one master computation node, and four slave nodes. The gateway node is the entry point to the infrastructure by the users and is also used for monitoring. The master node executes the head or master process for all tools that require such a head. The gateway and master nodes need considerably less CPU and memory resources than the slave nodes; in our setup we have mounted older and weaker servers on the rack for these nodes, allocating all four new servers as slave nodes.

All nodes are Virtual Machines (VM) deployed in separate physical machines. Each slave node uses two virtual drive images, one for the OS installation and one for data.

Rationale

This allows us to easily backup and restore the OS image or the data image in order to:

- Replace the OS installation image with a trusted OS image. This ensures that even if the computation nodes are compromised, any undetected back-doors are removed. This is applied periodically and after a breach is detected.
- Replace the OS installation image with an updated OS distribution to take advantage of new versions of the software bundled in the updated distribution.
- Backup and restore the data image, both for safety and to alternate between experiments that do not simultaneously fit in our storage.

We choose to use files instead of partitions as virtual drives because it is more complicated if you want to have the entire virtual machine contained in a partition. In that case, there would be no disk image file to actually boot the virtual machine since you cannot install a bootloader to a partition that is itself formatted as a file system and not as a partitioned device with a MBR. Such a virtual machine can be booted either by specifying the kernel and initrd manually, or by simulating a disk with a MBR by using linear RAID\(^1\).

\(^1\) https://wiki.archlinux.org/index.php/QEMU
2.1 Base Installation and Configuration

We used Debian 8 for all physical machines, as it is a stable and secure distribution. We assume a Debian 8 machine exposing only ssh to be trusted. The physical drive is partitioned as follows:

```
/    25 GB (RAID 1) this partition will host the OS
/srv  5934 GB (RAID 0) this partition will host the VM images
swap  8183 MB (RAID 1)
```

Virtual machines are executed on *Kernel-based Virtual Machine (KVM)* virtualization infrastructure\(^2\) for the Linux kernel. The following command installs the necessary Debian packages:

```
sh> sudo apt-get install postfix htop tmux sudo qemu-kvm libvirt-bin virtinst bridge-utils kpartx
```

**Rationale**

We only use the physical machines as hosts for our Virtual Machines so there is no need to always use the latest OS version. We only upgrade our machines to fix critical bugs or get security fixes.

2.2 Network Topology

Each physical machine that hosts the master and slaves VMs has four physical network interfaces, used as follows:

- eth0: management network
- eth1: computation network (br10)
- eth2: currently unused
- eth3: SAN network (br172)

The physical machine of the gateway node has 4 interfaces used as follows:

- eth0: management network
- eth1: computation network (br10)

eth2: public network (brDMZ)
eth3: SAN network (br172)

The gateway node VM has an external IP through the brDMZ bridge in order to be accessible from external networks at:

bdegw.iit.demokritos.gr or 143.233.226.33

The master and slave nodes are placed in a NAT and have cluster-internal IPs (br10). They are only accessible from the gateway node. The clusters network topology is depicted in Figure 1.

Rationale

By having only one node attached to the public network (brDMZ) and the rest of the nodes attached to the internal computation network (br10) we provide a secure isolation level to the cluster. This way the security of the computation network relies in keeping on node secure (gateway) which in general is easier and simpler to accomplish because one has to only keep one configured with optimal security options and also in case of a needed downtime for upgrades only one node has to be kept offline. By having a separate SAN network we provide access to consolidated block level data storage. The separate management network provides secure access for the management of the cluster.

2.3 Virtual Node Installation

We have used Ubuntu 16.04.3 for the master and slave node VMs, as Ubuntu regularly receives security fixes which can be installed as automatic updates. Version 16.04 is a Long Term Support release and will remain supported until April 2021.

We will use the native Ubuntu software repositories to install the following:

- General purpose tools: ntp, htop, nmon, wget, git, etc.
- iptables-persistent in order to give internet access to the slave nodes.
- nginx for accessing services from outside the cluster's NAT.

In the virt-install commands given below we use the /srv/vmimages directory to create and store the VM images. Any directory can be used, as long as it exists before the virt-install commands are executed.
2.3.1 Ansible

We use Ansible 2.3.2.0 to configure the VM nodes and install and configure the base components. To install Ansible follow the instructions provided in the Ansible documentation.\(^3\)

\(^3\) cf. http://docs.ansible.com/ansible/intro_installation.html
The BDE Ansible script is maintained in the big-data-europe/ansible Github repository. To checkout a local copy, issue:

```
sh> git clone https://github.com/big-data-europe/ansible.git
```

In the remainder of this document, we will write `<ANSIBLE_HOME>` to refer to the directory where the BDE Ansible script has been downloaded.

This script configures the network interfaces, installs auxiliary packages, deploys and configures the Docker tools. The script on the “master” branch of the ansible repository is always configured for the NCSR cluster. If your cluster is different edit the following files accordingly.

- `<ANSIBLE_HOME>/ansible.cfg` defines the username used for deployment.
- `<ANSIBLE_HOME>/hosts` configures the way your machine connects to the VMs over SSH and defines roles for the VMs (gateway, Swarm manager, Swarm worker).
- `<ANSIBLE_HOME>/files/hosts` defines the hostnames of the VMs in the cluster, it is used as the VMs hosts file.
- `<ANSIBLE_HOME>/files/bdegw_interfaces` configures the network interface that will be used by the gateway VM.
- `<ANSIBLE_HOME>/files/daemon.json` configures the Docker daemon of each VM.

### 2.3.2 Docker tooling

The Ansible playbook installs Docker engine and Docker Compose in every node that fills the Swarm worker role. This is all nodes except for the gateway node.

The Ansible playbook installs Docker engine and Docker Compose in every node that fills the Swarm manager role. The Swarm manager nodes are worker nodes specially configured as managers. In our setup managers are all nodes except for the gateway node.

---

**Rationale**

We moved from Docker Engine (v11) to Docker CE (v17) which introduces stability bug fixes. More importantly the new Docker version also introduces a new Swarm version, called Swarm mode. The new Swarm version introduces a simpler Swarm API to deploy the cluster and manage the nodes. It does not require an external Key-Value store and uses the Raft Consensus Algorithm to manage the global Swarm cluster state. It is more stable compared to the legacy Swarm where we were dealing with frequent fatal Swarm timeout issues when we had a large number of containers (more than 50) that all used the same network. Finally Swarm is more security oriented by design.

We decided to deploy Swarm Manager in every node. A small overhead is added for every node but Swarm has higher availability and a user can issue Swarm commands from every node.

**2.3.2 Gateway**

On the physical machine that will host the gateway VM execute the following command as root to create the VM:

```
```

Network configuration is as follows:

- **Hostname:** bdegw
- **Domain:** iit.demokritos.gr
- **Primary network interface:** eth1
- **IP:** 143.233.226.33
- **Netmask:** 255.255.255.128
- **Gateway:** 143.233.226.1
- **DNS:** 143.233.226.2 143.233.226.3

Disk partitioning:
/ vda1 20 GB
   swap vda2 all remaining space

Username: iitadmin

When the installer asks about automatic updates, choose "no"
When the installer asks to choose software to install select "openssh server"
When the installer asks about the default system clock choose "UTC"

This machine acts as the gateway to all other nodes (master and slaves). It is publicly accessible from

   bdegw.iit.demokritos.gr or 143.233.226.33

Before continuing with the installation of the other nodes, the bdegw node must be configured so that the other nodes have Internet access.

For security reasons bdegw is accessible from ssh on port 222. The first thing you have to do is to change ssh port from default 22 to 222. To do so login to the physical machine that hosts the gateway VM bdegw and then connect to it with

   sh> sudo virsh console bdegw

Login and then run the following commands to change the ssh port

   sh> ssh iitadmin@bdegw.iit.demokritos.gr
   sh> sudo sed -i 's/Port 22/Port 222/g' /etc/ssh/sshd_config
   sh> sudo /etc/init.d/ssh restart

Then setup password-less ssh access from your machine by issuing:

   sh> ssh -p222 iitadmin@bdegw.iit.demokritos.gr 'cat >> ~/.ssh/authorized_keys' < ~/.ssh/id_*.pub

After that configure bdegw with the ansible script by running:

   sh> git clone https://github.com/big-data-europe/ansible.git
   sh> cd ansible
   sh> ansible-playbook playbook.yaml -i hosts --limit gateway -K -vvv

When prompted for SUDO password enter the password for "bdegw".
After the scripts finishes you are ready to configure the rest of the VMs.

2.3.3 Master node
On physical machine that will host the master node VM execute the following command as root to create VM:

```
sh> virt-install -n master --ram 3900 --vcpus=8 --os-type linux --disk
    path=/srv/vmimages/master.img,device=disk,bus=virtio,sparse=true,format=raw,size=300 --location
    'http://de.archive.ubuntu.com/ubuntu/dists/xenial/main/installer-amd64/' --graphics none --
    accelerate --network bridge:br10,model=virtio --extra-args 'console=ttyS0,115200n8 serial' --
    autostart
```

Network configuration is as follows:

- Hostname: master
- Domain name: bde-cluster
- IP Address: 10.0.10.10
- Netmask: 255.255.255.0
- Gateway: 10.0.10.1
- DNS: 143.233.226.2 143.233.226.3

Disk partitioning:
- /: vda1 290 GB
- swap: vda2 all remaining space

Username: iitadmin

When the installer asks about automatic updates, choose “no”

When the installer asks to choose software to install select “openssh server”

When the installer asks about the default system clock choose “UTC”

2.3.4 Slave nodes
On the slave physical machines, execute the following command as root to create the slave node VM images:
sh> virt-install -n
<VM_NAME> --ram 30000 --vcpus=4 --os-type linux --disk
path=/srv/vmimages/<VM_NAME>.img,device=disk,bus=virtio,sparse=true,
format=raw,size=25 --disk
path=/srv/vmimages/<VM_NAME>_data.img,device=disk,bus=virtio,sparse=true,
format=raw,size=2500
--location http://de.archive.ubuntu.com/ubuntu/dists/xenial/main/installer-amd64/
--graphics none --accelerate --network bridge:br10,model=virtio --extra-args
'console=ttys0,115200n8 serial' --autostart

Replacing <VM_NAME> with slave1, slave2, etc.

Network configuration is as follows:

- Hostname: slave1
- Domain name: bde-cluster
- IP Address: 10.0.10.11
- Netmask: 255.255.255.0
- Gateway: 10.0.10.1
- DNS: 143.233.226.2 143.233.226.3

Disk partitioning: The / partition contains all system libraries and executables. The /srv partition contains data. This way the base system can be updated or replaced by just replacing the vda1 image without having to copy the data stored in the vdb1 image.

- / vda1 20 GB
- swap vda2 all remaining space
- /srv vdb1 whole disk

Username: iitadmin

When the installer asks about automatic updates, choose "no"

When the installer asks to choose software to install select "openssh server"

When the installer asks about the default system clock choose "UTC"

2.3.5 Master and slave node configuration

All the VMs are configured using the Ansible script as described in the previous section.
To run the script you must first setup password-less SSH access to the nodes. To do so follow the instructions provided in the “2.8 BDE Cluster SSH Access” section of this document. Then execute the ansible script:

```
sh> git clone https://github.com/big-data-europe/ansible.git
sh> cd ansible
sh> ansible-playbook playbook.yaml -i hosts --limit swarm_nodes -K -vvv
```

When prompted for SUDO password enter the password for master and slaves.

After ansible finishes a Docker Swarm cluster should be deployed in all nodes. To verify the installation login in any master or slave node and execute:

```
sh> docker node ls
```

If Swarm was configured correctly the above command will list all nodes in the Swarm cluster.

### 2.4 Recovery from Failures

In this section we describe how to recover from failures such as a VM crash or a hardware failure. There are two scenarios, a failure on the master node or a failure on slave node.

**2.4.1 Master node failure**

If the master node stops then login in the physical machine that hosts the master VM and start the VM using:

```
sh> sudo virsh start master
```

In case of hardware failure then you should restore the setup from RAID1 and follow the steps above to restart the VM and services.

**2.4.2 Slave node failure**

If a slave node stops then login in its host machine and start it using:

```
sh> sudo virsh start <VM_NAME>
```

To recover from a hardware failure you should first start by setting up the new physical machine using steps described above (Sections 2.1 and 2.3). Then login to the physical
machine and create the VM. In this scenario we will re-create slave1 with IP 10.0.10.11; replace the name and IP as appropriate:

```
sh> virt-install -n slave1 --ram 8192 --vcpus=4 --os-type linux --disk path=/srv/vmimages/slave1.img,device=disk,bus=virtio,sparse=true,format=raw,size=25 --disk path=/srv/vmimages/slave1_data.img,device=disk,bus=virtio,sparse=true,format=raw,size=2500 --location 'http://de.archive.ubuntu.com/ubuntu/dists/xenial/main/installer-amd64/' --graphics none --accelerate --network bridge:br10,model=virtio --extra-args 'console=ttyS0,115200n8 serial' --autostart
```

Disk partitioning and network configuration is as described above. After finishing the installation, proceed to configure with Ansible. To run the Ansible script you must first setup password-less SSH access to the node. To do so follow the instructions provided in the “2.8 BDE Cluster SSH Access” section of this document.

To configure the node and add it to Docker Swarm run:

```
sh> cd <ANSIBLE_HOME>
sh> ansible-playbook playbook.yaml -i hosts --limit slave1.bde-cluster -K -vvv
```

When prompted for SUDO password enter the password for “slave1”.

To verify that the node was added correctly in the Swarm cluster login in any master or slave node and execute:

```
sh> docker node ls
```

If Swarm was configured correctly the above command will list all nodes in the Swarm cluster.

### 2.5 Adding a New Node to a Live Swarm

In this section we describe how to add a new node in the cluster. It is important to note that these instructions refer to adding a new node without restarting Swarm or affecting in any way computations already executing at the rest of the cluster.

First you have to create a new VM. In the physical machine create a new VM with KVM and assign a new name e.g. slave5 with IP 10.0.10.15.

To create the VM run on the physical machine:
sh> virt-install -n slave5 --ram 8192 --vcpus=4 --os-type linux --disk
path=/srv/vmimages/slave5.img,device=disk,bus=virtio,sparse=true,format=raw,size=25 --disk
path=/srv/vmimages/slave5_data.img,device=disk,bus=virtio,sparse=true,format=raw,size=2500 --location 'http://de.archive.ubuntu.com/ubuntu/dists/xenial/main/installer-amd64/' --graphics none --accelerate --network bridge:br10,model=virtio --extra-args 'console=ttys0,115200n8 serial' --autostart

Installation-time choices, network configuration and disk partition are as in Section 2.5.2 above.

Add all nodes (bdegw, master and slaves) to /etc/hosts of all nodes:

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Machine Name</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.10.10</td>
<td>master.bde-cluster</td>
<td>master</td>
</tr>
<tr>
<td>10.0.10.11</td>
<td>slave1.bde-cluster</td>
<td>slave1</td>
</tr>
<tr>
<td>10.0.10.12</td>
<td>slave2.bde-cluster</td>
<td>slave2</td>
</tr>
<tr>
<td>10.0.10.13</td>
<td>slave3.bde-cluster</td>
<td>slave3</td>
</tr>
<tr>
<td>10.0.10.14</td>
<td>slave4.bde-cluster</td>
<td>slave4</td>
</tr>
<tr>
<td>10.0.10.15</td>
<td>slave5.bde-cluster</td>
<td>slave5</td>
</tr>
</tbody>
</table>

In the <ANSIBLE_HOME>/files/hosts add line:

```
10.0.10.15 slave5.bde-cluster slave5
```

as above. All machines in the cluster must be listed there so modify the /etc/hosts file accordingly.

In the <ANSIBLE_HOME>/hosts add the following line under section "swarm_managers" if the new node will be a Swarm manager or under "swarm_workers" you are adding a new Swarm worker

```
slave5.bde-cluster ansible_ssh_port=22 ansible_ssh_user=iitadmin
```

The new node should be also added in your ~/.ssh/config file and have passwordless SSH enabled as described in section 2.8 BDE cluster SSH access.

Then run the ansible script to configure the new node and add it to Docker Swarm

```
sh> cd <ANSIBLE_HOME>
sh> ansible-playbook playbook.yaml -i hosts --limit slave5.bde-cluster -K -vvv
```

When prompted for SUDO password enter the password for "slave5".
To verify that the node was added correctly in the Swarm cluster login in any master or slave node and execute:

```
sh> docker node ls
```

If Swarm was configured correctly the above command will list all nodes in the Swarm cluster.

### 2.6 Cloning a Node

Instead of creating a new VM by following the typical installation procedure one can alternatively clone another VM and use it as a new node of the cluster. To do this the VM to be cloned has to be unchanged since its typical installation was completed and not configured with Ansible.

Use the following commands as root to clone a VM. In the following example we clone slave1 to slave2. Replace the names accordingly for a different scenario (e.g. slave2 to slave3)

First you must shutdown the VM you want to clone using

```
sh> virsh shutdown slave1
```

Then copy original images to cloned. Use sparse=always to copy only the real size and not the virtual disk space reserved by QEMU:

```
sh> cp -p --sparse=always /srv/vmimages/slave1.img:q /srv/vmimages/slave2.img
sh> cp -p --sparse=always /srv/vmimages/slave1_data.img /srv/vmimages/slave2_data.img
```

Clone the KVM XML definitions

```
sh> virt-clone -o slave1 -n slave2 --file=/srv/vmimages/slave2.img --file=/srv/vmimages/slave2_data.img --preserve-data
```

Mount the images to edit the network configuration

```
sh> losetup /dev/loop0 /srv/vmimages/slave2.img
sh> kpartx -a /dev/loop0
sh> mount /dev/mapper/loop0p1 /mnt/
```

Change the hostname

```
sh> echo slave2 > /mnt/etc/hostname
```
ensure that the following records exist to the /mnt/etc/hosts file otherwise add the missing records.

<table>
<thead>
<tr>
<th>IP Address</th>
<th>VM Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.10.1</td>
<td>bdegw.bde-cluster</td>
</tr>
<tr>
<td>10.0.10.10</td>
<td>master.bde-cluster</td>
</tr>
<tr>
<td>10.0.10.11</td>
<td>slave1.bde-cluster</td>
</tr>
<tr>
<td>10.0.10.12</td>
<td>slave2.bde-cluster</td>
</tr>
<tr>
<td>10.0.10.13</td>
<td>slave3.bde-cluster</td>
</tr>
<tr>
<td>10.0.10.14</td>
<td>slave4.bde-cluster</td>
</tr>
</tbody>
</table>

To change the ip address of the cloned VM edit file /mnt/etc/network/interfaces and replace line "address" with the IP of the new VM. e.g. for slave2 replace address 10.0.10.11 with address 10.0.10.12

Unmount the images, free up the resources, and start the VM:

sh> umount /mnt
sh> kpartx -d /dev/loop0
sh> losetup -d /dev/loop0
sh> virsh start slave2

2.7 Pilot Swap

In this scenario the user wants to swap between pilots and keep the data of the previous pilot. To do so follow the steps below.

First shutdown the master and slave VMs using

sh> virsh shutdown <VMname>

where <VMname> the name of the VM in all host machines starting from the master. Then replace all the images of the stopped VMs with the images the new pilot is using. Then startup all VMs by using

sh> virsh start <VMname>

Always keep backup of the replaced images.
For example to replace the master VM log in the physical machine that hosts the master VM and run

```bash
sh> virsh shutdown master
```

Then if you have an image for the master of the new pilot in `/srv/vmimages/master_pilot2.img` first take a backup of the running pilot 1 and then replace the image with pilot 2 image by running

```bash
sh> mv /srv/vmimages/master.img /srv/vmimages/master_pilot1.img
sh> mv /srv/vmimages/master_pilot2.img /srv/vmimages/master.img
```

After the `mv` command finishes start-up the VM using

```bash
sh> virsh start master
```

Note: You should always first replace all the images of master and slaves and then startup the VMs starting from the master VM.

### 2.8 BDE Cluster SSH Access

To run the Ansible scripts you must first configure SSH and then enable password-less authentication on each VM you create. The same procedure is proposed for accessing the cluster with SSH.

First configure your SSH client through the `~/.ssh/config` file. If directory `~/.ssh` and file `~/.ssh/config` do not exist you must create them. Then add the following configuration to the `config` file and you will be able to connect to the private hosts with one command. Modify the file to match the machines in your cluster and replace “iitadmin” with the name of the remote user if it is different.

```bash
Host bdegw
  HostName bdegw.iit.demokritos.gr
  User iitadmin
  Port 222

Host master.bde-cluster
  User iitadmin
  ProxyCommand ssh -q bdegw nc -q0 master.bde-cluster 22
```
Host slave1.bde-cluster
User iitadmin
ProxyCommand ssh -q bdegw nc -q0 slave1.bde-cluster 22
Host slave2.bde-cluster
User iitadmin
ProxyCommand ssh -q bdegw nc -q0 slave2.bde-cluster 22
Host slave3.bde-cluster
User iitadmin
ProxyCommand ssh -q bdegw nc -q0 slave3.bde-cluster 22
Host slave4.bde-cluster
User iitadmin
ProxyCommand ssh -q bdegw nc -q0 slave4.bde-cluster 22

After that you should be able to connect for example to slave1.bde-cluster machine by issuing

sh> ssh slave1.bde-cluster

Then exit slave1 and enable password-less authentication in all VMs. If you don’t already have a SSH keypair create one in your machine with

sh> ssh-keygen

First you must configure the “bdegw” for SSH password-less authentication if you haven’t already done this in a previous step. To do so issue from your machine

sh> ssh -p222 iitadmin@bdegw.iit.demokritos.gr 'cat >> ~/.ssh/authorized_keys' < ~/.ssh/id_*.pub

Then enable password-less authentication in the rest of the machines by running from your machine

sh> ssh-copy-id <host>

Where <host> the host you want to configure. For example to enable password-less authentication for slave1 run

sh> ssh-copy-id slave1.bde-cluster
Alternatively you can connect to the cluster without configuring SSH. To access a machine within the cluster you must first ssh to the cluster gateway bdegw.iit.demokritos.gr (143.233.226.33) and from there ssh to the internal machine of your choice.

For example if you want to connect to master.bde-cluster as user iitadmin you can run the following command

```bash
sh> ssh -A -t -p222 iitadmin@bdegw.iit.demokritos.gr ssh -A -t 10.0.10.10
```

which combines two SSH commands, one to connect to bdegw and from there to the master node.

### 2.9 Port Forwarding

The base installation also deploys a nginx server on the bdegw node. By using nginx you can make ports and UIs available through the public bdegw.iit.demokritos.gr URL.

To do so you must add a server section in file `/etc/nginx/conf.d/tcp_forwarding.conf` on bdegw node like below

```plaintext
server {
    listen 143.233.226.33:<EXPOSED_PORT>;
    location / {
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass http://<HOST>:<PORT>;
    }
}
```

Replace `<PORT>` and `<HOST>` with those you want to forward and `<EXPOSED_PORT>` with the port that will be publicly available. Then you must restart nginx with

```bash
sh> sudo service nginx restart
```

After that you can access the service from bdegw.iit.demokritos.gr:<PORT>.
For example if you want to expose the Web UI running on 10.0.10.1:11111 the server section should be the following

server {
    listen 143.233.226.33:11111;

    location / {
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass http://10.0.10.1:11111;
    }
}

Then issue

    sh> sudo service nginx restart

After that you can access the GUI from bdegw.iit.demokritos.gr:11111.

Another way to temporarily forward ports is to use ssh tunneling. To do so from your machine run

    sh> ssh -L localhost:<local_port>:<VM_IP>:<VM_port> -p222 iitadmin@bdegw.iit.demokritos.gr

and replace <local_port> with the local port you are going to use, <VM_IP> the IP of the node you want to access and <VM_port> the port in the VM you want to access. For example if you want to access a service listening on port 9042 in slave4 from localhost:9042 run

    sh> ssh -L localhost:9042:10.0.10.14:9042 -p222 iitadmin@bdegw.iit.demokritos.gr

3. Other BDE components

Although the components of a pipeline are all contained in the docker-compose.yml specifying the pipeline, components can be started manually for testing purposes. In this
section we describe how to deploy BDE components. All containers can be deployed in any node. You can define the node by setting the node’s hostname on the “node.hostname” constraint. For example to deploy Semagrow on “slave1” the docker-compose.yml file should be:

```yaml
version: "3"

services:

  semagrow:
    image: semagrow/semagrow
    deploy:
      placement:
        constraints: [node.hostname == slave1]
```

All Docker Swarm commands can be executed from any Swarm manager node. To deploy a stack named “example-stack” described in the “docker-compose.yml” file issue the following:

```bash
sh> docker stack deploy --compose-file docker-compose.yml example-stack
```

More info about each container can be found at BDE wiki\(^5\).

### 3.1 Apache Spark

To deploy a standalone Spark cluster with one Spark master running in node “master” of our example setup and multiple Spark workers running in every other node use the following docker compose snippet.

```yaml
version: "3"

services:
```

spark-master:
  image: bde2020/spark-master:2.2.0-hadoop2.7
  hostname: spark-master
  deploy:
    placement:
      constraints: [node.hostname == master]

spark-worker:
  image: bde2020/spark-worker:2.2.0-hadoop2.7
  deploy:
    mode: global
    placement:
      constraints: [node.hostname != master]
  depends_on:
    - spark-master

### 3.2 OpenLink Virtuoso

To deploy a Virtuoso container use the following docker compose snippet

```yaml
version: "3"

services:

  virtuoso:
    image: tenforce/virtuoso
```
3.3 Semagrow and sevod-scraper

In order to use Semagrow you must first provide a sevod\(^6\) description for the federated datasets. If you want to create one you can use the docker container for the sevod-scraper tool. The tool creates a sevod description both for a triple store and Cassandra.

To create sevod description from a RDF dump run

```
sh> docker run --rm -it -v /path/to/output:/output -v /path/to/dump:/dump semagrow/sevod-scraper rdfdump /dump/<dump_name> <endpoint_url> <flags> /output/<output_name>
```

Where:
- `/path/to/output` the directory to write the output
- `/path/to/dump` the directory that contains the dump
- `dump_name` the filename of the dump
- `endpoint_url` the endpoint URL where the dump is stored
- `flags` the flags to run sevod-scraper
- `output_name` the filename of the output which will be located at `/path/to/output/output_name`

To create sevod description from Cassandra run

```
sh> docker run --rm -it -v /path/to/output:/output -v semagrow/sevod-scraper cassandra <cassandra_ip> <cassandra_port> <keyspace> <base_url> /output/<output_name>
```

Where:
- `/path/to/output` the directory to write the output
- `cassandra_ip` the IP of the Cassandra store
- `cassandra_port` the port of the Cassandra store
- `keyspace` the Cassandra keyspace to scrap
- `base_url` the base url to use in the output
- `output_name` the filename of the output which will be located at `/path/to/output/output_name`

To deploy a Semagrow Docker container issue

```
sh> docker run -d -v /path/to/sevod:/etc/default/semagrow semagrow/semagrow
```

\(^6\) https://www.w3.org/2015/03/sevod
To deploy Semagrow with the Cassandra extension run

```
sh> docker run -d -v /path/to/sevod:/etc/default/semagrow semagrow/semagrow-cassandra
```

### 3.4 4store

To deploy a 4store cluster you must first deploy all datanode containers in the slave nodes and then the 4store master in the master node. To deploy the stack use the following docker compose snippet

```
version: "3"

services:

  4store1:
    image: bde2020/4store
    deploy:
      placement:
        constraints: [node.hostname == slave1]

  4store2:
    image: bde2020/4store
    deploy:
      placement:
        constraints: [node.hostname == slave2]

  4store3:
    image: bde2020/4store
    hostname: 4store1
    deploy:
```

Page 31
3.5 GeoTriples

To deploy GeoTriples use the following docker compose snippet

```yaml
version: "3"

services:
```
geotriples:
    image: bde2020/geotriples

3.6 Sextant
To deploy Sextant use the following docker compose snippet

```yaml
version: '3'

services:

  sextant:
    image: bde2020/sextant
    deploy:
      placement:
        constraints: [node.hostname == master]
    ports:
      - "8890:8890"
```

If you want to make Sextant’s UI publicly available you must forward the UI through the bdegw node. To do so log into bdegw and add in file `/etc/nginx/conf.d/tcp_forwarding.conf` the following section

```bash
server {
  listen 143.233.226.33:9999;
  location / {
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-Server $host;
  }
}
```
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_pass http://10.0.10.10:8890/;
}
}

Replace 143.233.226.33:9999 with the public IP and port that will be used and 10.0.10.10 with the IP of the node where Sextant will be deployed.

Then restart nginx with:

```
sh> sudo service nginx restart
```

The Sextant interface will now be available at bdegw.iit.demokritos.gr:9999.

3.7 Silk

To deploy Sextant for example in the master use the following docker compose snippet:

```
version: '3'

services:

silk:
  image: bde2020/silk-workbench
  deploy:
    placement:
      constraints: [node.hostname == master]
    ports:
      - "8890:8890"
```

If you want to make Silk’s UI publicly available you must forward the UI through the bdegw node. To do so log into bdegw and add in file `/etc/nginx/conf.d/tcp_forwarding.conf` the following section:
server {
  listen 143.233.226.33:9000;

  location / {
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-Server $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_pass http://10.0.10.10:8890/;
  }
}

Replace 143.233.226.33:9999 with the public IP and port that will be used and 10.0.10.10 with the IP of the node where Sextant will be deployed.

Then restart nginx with

    sh> sudo service nginx restart

The Silk interface will now be available at bdegw.iit.demokritos.gr:9000

3.8 PostGIS

To deploy PostGIS use the following docker compose snippet

    version: '3'

    services:

        postgis:

            image:bde2020/postgis
3.9 Strabon
To deploy Strabon you must also deploy PostGIS. To do so use the following docker compose snippet

```yaml
version: '3'

services:

  strabon:
    image: bde2020/strabon

  postgis:
    image: bde2020/postgis
```

3.10 Cassandra
To deploy Cassandra for example you should use the following docker compose snippet

```yaml
version: '3'

services:

  cassandra:
    image: bde2020/cassandra
```

To deploy a Cassandra cluster on slave nodes use the following docker compose snippet

```yaml
version: "3"

services:

  cassandra-cluster-1:
```
image: bde2020/cassandra
environment:
  - "CASSANDRA_BROADCAST_ADDRESS=cassandra-cluster-1"

deploy:
  placement:
    constraints: [node.hostname == slave1]

cassandra-cluster-2:
image: bde2020/cassandra
environment:
  - "CASSANDRA_BROADCAST_ADDRESS=cassandra-cluster-2"
  - "CASSANDRA_SEEDS=cassandra-cluster-1"

deploy:
  placement:
    constraints: [node.hostname == slave2]
  depends_on:
    - cassandra-cluster-1

cassandra-cluster-3:
image: bde2020/cassandra
environment:
  - "CASSANDRA_BROADCAST_ADDRESS=cassandra-cluster-3"
  - "CASSANDRA_SEEDS=cassandra-cluster-1"

deploy:
  placement:
    constraints: [node.hostname == slave3]
depends_on:
  - cassandra-cluster-1

cassandra-cluster-4:
  image: bde2020/cassandra
  environment:
    - "CASSANDRA_BROADCAST_ADDRESS=cassandra-cluster-4"
    - "CASSANDRA_SEEDS=cassandra-cluster-1"
  deploy:
    placement:
      constraints: [node.hostname == slave4]
    depends_on:
      - cassandra-cluster-1

3.11 Hadoop
To deploy a Hadoop cluster with the management nodes running on master and the datanodes on the slaves you must first clone the docker-hadoop repository and go into the clone directory by running on any node

sh> git clone https://github.com/big-data-europe/docker-hadoop.git
sh> cd docker-hadoop

Then use the following docker compose snippet

version: '3'

services:
	namenode:
    image: bde2020/hadoop-namenode:2.0.0-hadoop2.7.4-java8
environment:
  - CLUSTER_NAME=test

defploy:
  mode: replicated
  replicas: 1
  restart_policy:
    condition: on-failure
  placement:
    constraints:
    - node.hostname == master

resourcemanager:
  image: bde2020/hadoop-resourcemanager:2.0.0-hadoop2.7.4-java8

resourcemanager:
  image: bde2020/hadoop-datanode:2.0.0-hadoop2.7.4-java8
environment:
   SERVICE_PRECONDITION: "namenode:50070 datanode:50075"
env_file:
   - ./hadoop.env
deploy:
   mode: replicated
   replicas: 1
   restart_policy:
      condition: on-failure
placement:
   constraints:
      - node.hostname == master
healthcheck:
   disable: true

nodemanager:
   image: bde2020/hadoop-nodemanager:2.0.0-hadoop2.7.4-java8
   environment:
      SERVICE_PRECONDITION: "namenode:50070 datanode:50075 resourcemanager:8088"
   env_file:
      - ./hadoop.env
deploy:
   mode: global
   restart_policy:
      condition: on-failure
historyserver:
  image: bde2020/hadoop-historyserver:2.0.0-hadoop2.7.4-java8
  environment:
    SERVICE_PRECONDITION: "namenode:50070 datanode:50075 resourcemanager:8088"
  env_file:
    - ./hadoop.env
  deploy:
    mode: replicated
    replicas: 1
    placement:
      constraints:
        - node.hostname == master

3.12 Flink
To deploy Flink master at master node and Flink workers in all other nodes use the following docker compose snippet

    version: "3"

services:

  flink-master:
    image: bde2020/flink-master
    hostname: flink-master
    deploy:
      placement:
        constraints: [node.hostname == master]
flink-worker:
  image: bde2020/flink-worker
  #hostname: flink-worker
  environment:
    - FLINK_MASTER_PORT_6123_TCP_ADDR=flink-master
  deploy:
    mode: global
    placement:
      constraints: [node.hostname != master]
  depends_on:
    - flink-master
4. Installing and Executing BDI Stacks

In the following subsections we will describe how to install, launch and monitor an existing pilot stack in the BDI. We assume the BDI platform is installed. For building a stack from scratch without the reuse of existing BDI prepared components, we refer to the more detailed documentation.

4.1 Installing and Launching the BDI IDE

To install the BDI IDE, run the following commands in the terminal:

> git clone https://github.com/big-data-europe/app-bdi-ide.git
> cd app-bdi-ide
> sudo ./scripts/install.sh

The installation script will configure the BDI application and set up the links correctly in the /etc/hosts file so that the addresses used by the BDI point to localhost.
The following command will launch the BDI (assuming that the current directory is `app-bdi-ide/` or the directory specified when cloning):

```bash
> docker-compose up -d
```

To log a specific service, type the following, or omit [SERVICE] to monitor the entire BDI:

```bash
> docker-compose logs -f [SERVICE]
```

### 4.2 Installing a Stack on the BDI

An execution stack should follow a structure similar to that of the BDE pilots, which are available from https://github.com/big-data-europe/ and can be used as starting points to accelerate development. In any case, it should be the case that:

- There is a GitHub repository where the pilot stack is stored
- The pilot stack does not use ports or service names used by the BDI core tools.
- The network topology is as described above.

Steps to be executed:

1. Go to the Github page of the pilot stack
2. Copy the clone URL
3. Go to the BDI
4. Click on Swarm UI
5. Click on 'create a new stack'
6. Copy the URL from the clipboard, fill in a [STACK_NAME] and optionally an icon
7. Click save

If there are conflicts over the usage of ports, names or a custom network:

1. click on [STACK_NAME]
2. click on the edit docker compose file
3. fix the issues
4. click on save
BDE Pipeline Monitor

Sensor demo

Steps

- Setup HDFS
  - Booting of the HDFS cluster.

- Setup SPARK
  - Starts the spark execution and workers.

- Populate HDFS with core data
  - Please upload the location data to the HDFS filesystem. This is a manual step. Press finish when you're done.

- Aggregate location data

Dashboard

Stack Builder

Stack Editor
4.3 Launching the stack

The BDE Stack includes a workflow monitor with similar functionality to Docker’s Health Check (which did not exist at the time that the BDE architecture was decided on). There are two approaches to enabling workflow monitoring:

1. The BDI offers a workflow monitor service (the init daemon) that can be used by platforms running on the BDI. This requires joining the init daemon to the platform’s network.
2. Another approach is for the pilot stack itself to contain an init daemon and handle the pipeline process itself.

The advantage of the first approach is that more than one stack can depend on a certain step, but with the additional requirement that all necessary data be present in the BDI SPARQL endpoint (http://localhost:8890). The second approach allows for a complete plug-and-play setup of the pilot stack on the BDI.

Launching a stack:

1. Go to the swarm-admin
2. Click on ‘Stacks’ and open the desired stack
3. Click on launch pipeline
4. [automatic] This will transfer you to the pipeline view
5. Click on “UP”

4.4 Monitoring a stack

Every stack can have its own workflow. In general some steps will be automatic, while others, such as uploading datasets, will be manual. For more details, please refer to the Pilot Documentation.

To monitor the services, during installation (Section 4.1), the following extra steps are also necessary:

1. click on edit docker compose file
2. add a label "logging=true" for each service for which logging is desired
3. click on save

During execution, you can then:

1. Click on "NetLog"
2. Add "*har" to the pattern field
3. Click on "create"
4. Click on dashboard

If necessary, it is possible to build additional custom dashboards and visualisation with more specific information.
5. Conclusion

The base platform has been deployed on the NCSR infrastructure. The installation procedure was independently tested. The cluster as installed here will be used as a platform
for testing the SC implementations. This infrastructure will continue to be updated as necessary.

5.1 Previous Deployment Testing

In the table below we present records taken during the first round of deployment tests (command count to deploy the platform and time to deploy) with three subjects, two of them not working for BDE and not familiar with the platform:

1. G. Stavrinos: Expert Linux user; part of the NCSR-D BDE-tech team; also subject for earlier BDE installation experiments.
2. G. Siantikos: Expert Linux user; not part of the NCSR-D BDE team; research and development background in signal analysis; has never used or installed any Apache Big Data tools previously.
3. D. Sgouropoulos: Expert Linux user; not part of the NCSR-D BDE team; research and development background in machine learning; has never used or installed any Apache Big Data tools previously.

The first subject (G. Stavrinos) issued more commands and took more time to deploy the platform because he was the first to test it and provided comments and corrections on the procedure. The instructions were then updated and the two last subjects completed the guide with less commands and in less time.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Command count</th>
<th>Time to deploy</th>
</tr>
</thead>
<tbody>
<tr>
<td>G. Stavrinos</td>
<td>60</td>
<td>1h55m</td>
</tr>
<tr>
<td>G. Siantikos</td>
<td>56</td>
<td>1h22m</td>
</tr>
<tr>
<td>D. Sgouropoulos</td>
<td>56</td>
<td>1h26m</td>
</tr>
</tbody>
</table>

5.1 Current Deployment Testing

In the table below we present records taken during the second deployment tests with three subjects, one that participated in the first test deployments and two new subjects, one Linux expert user and one novice user.
1. G. Stavrinos: Expert Linux user; part of the NCSR-D BDE-tech team; also subject for earlier BDE installation experiments.
2. A. Davvetas: Expert Linux user; part of the NCSR-D BDE-SC5 team; research and development background in Machine Learning; has never used or installed any Apache Big Data tools previously; developer for SC5.
3. I. Christou: Novice Linux user; not part of the NCSR-D BDE team; research and development background in Physics; has never used or installed any Apache Big Data tools previously.

The first subject (G. Stavrinos) issued more commands and took more time to deploy the platform because he was the first to test it and provided comments and corrections on the procedure.

The first and second subjects took less time to deploy than the previous test deployments mainly because of the updated Ansible script. The script now runs faster, with less restarts and waiting times. Also it deploys a the latest version of Docker Swarm with does not require the download of extra containers to set up the Swarm and also has improved communication between the Swarm nodes so the cluster is set up faster.

The third subject took more time to deploy because he has less experience with Linux systems and container technology and took more time to understand the concepts and actions needed. He provided useful input on how to make the deployment and this guide more user friendly to novice Linux users.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Command count</th>
<th>Time to deploy</th>
</tr>
</thead>
<tbody>
<tr>
<td>G. Stavrinos</td>
<td>56</td>
<td>1h02m</td>
</tr>
<tr>
<td>A. Davvetas</td>
<td>56</td>
<td>1h07m</td>
</tr>
<tr>
<td>I. Christou</td>
<td>56</td>
<td>1h30m</td>
</tr>
</tbody>
</table>