VOSpace administrator guide

Issue/Rev. No.: 1.0
Date: March 24, 2023

Authors: Cristiano Urban[a], Sara Bertocco[a], Nicola Fulvio Calabria[a], Cristina Knapic[a], Riccardo Smareglia[a], Massimo Sponza[a] and Sonia Zorba
Affiliations: [a]INAF - Istituto Nazionale di Astrofisica - Osservatorio Astronomico di Trieste

Approved by: Cristina Knapic

Contents

1 Introduction

In this technical report we want to describe the underlying architecture of the INAF VOSpace from the system administrator point of view.
In the next sections we will illustrate the characteristics of the hosts, the management of the services, the installation and deployment of the VOSpace and other important administration topics.

2 VOSpace infrastructure

The current VOSpace infrastructure (see Figure 1) is composed by four hosts: `tn-bastion.ia2`, `vospace.ia2`, `vospace-tn.ia2` and `db02.ia2`.
The VOSpace consists of four services: VOSpace UI and VOSpace REST running on the frontend `vospace.ia2`; VOSpace File Service and VOSpace Transfer Service running on the backend `vospace-tn.ia2`.
The VOSpace database is hosted on the physical node `db02.ia2` to guarantee better performance.
The `tn-bastion.ia2` is a general purpose transfer node mainly used for huge amounts of data. Users can access it upon request using SSH and their INAF credentials.
The backend shares data with the transfer node and a tape library frontend machine respectively through Lustre and Spectrum Scale (GPFS) mount points.
A typical use case is the one where a user wants to retrieve some cold data previously stored into the VOSpace. The user launches a job through the VOSpace UI. The VOSpace UI sends a request to the VOSpace REST service which in turn creates a new asynchronous job. The job is queued and then executed by the VOSpace Transfer Service. Once the process is complete, data is available to download using the VOSpace UI (which leans on the VOSpace File Service) or, alternatively, using the transfer node (e.g. with rsync over SSH).

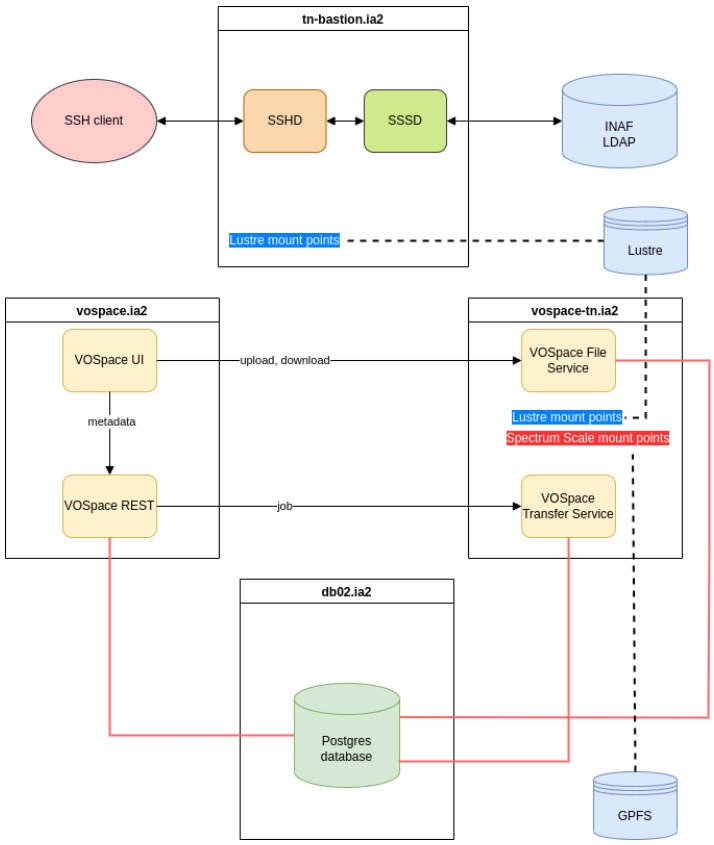

Figure 1: The INAF VOSpace infrastructure.

2.1 Frontend

Name	vospace.ia2
CPU(s)	2
RAM	8 GB
VIRTUAL DISKS	3
/boot (1 GB) / (20 GB) swap (4 GB)	
OS	CentOS 7

This VM hosts the **VOSpace UI** and **VOSpace REST** services. These services are placed behind a reverse proxy using the Apache http server.
The proxy configuration is stored in `/etc/httpd/conf.d/vospace-proxy.conf` and contains the following directives:

```
ProxyPreserveHost On

ProxyPass "/ui" "http://localhost:8085/ui" nocanon
ProxyPassReverse "/ui" "http://localhost:8085/ui"

ProxyPass "/" "http://localhost:8083/" nocanon
ProxyPassReverse "/" "http://localhost:8083/"

# Redirect root to /ui
RewriteEngine On
RedirectMatch ^/$ /ui
```

The VOSpace UI can be reached at the following URL: http://vospace.ia2.inaf.it/ui/.
The list of nodes is available at the following URL: http://vospace.ia2.inaf.it/nodes.

2.2 Backend

Name	vospace-tn.ia2
CPU(s)	2
RAM	8 GB
VIRTUAL DISKS	3
/boot (1 GB) / (20 GB) swap (4 GB)	
OS	CentOS 7

This VM hosts the **VOSpace File Service** and the **VOSpace Transfer Service**.

2.3 Database node

Name	db02.ia2
CPU(s)	64
RAM	256 GB
PARTITIONS (the most relevant...)	
/boot (1 GB) / (50 GB) /home (500 GB) swap (4 GB)	
OS	CentOS 7

This node hosts the **vospace** PostgreSQL database which contains the following tables:

```
[root@db02 ~]# su - postgres
-bash-4.2$ psql vospace -c "\dt"
             List of relations
 Schema |      Name       | Type  |  Owner
--------+-----------------+-------+----------
 public | deleted_node    | table | vospace
 public | job             | table | vospace
 public | linked_service  | table | vospace
 public | location        | table | vospace
 public | node            | table | vospace
 public | storage         | table | vospace
 public | users           | table | vospace
(7 rows)
```

The following query returns all the VOSpace storage points:

```
[root@db02 ~]# su - postgres
-bash-4.2$ psql vospace -c "select * from storage"
 storage_id | storage_type |                base_path                 |      hostname
------------+--------------+------------------------------------------+---------------------
          3 | hot          | /mnt/hot_storage/users                   | localhost
          4 | cold         | /ia2_tape_stb_01/users                   | tape-fe.ia2
          1 | local        | /lustre/ia2-lst1/vospace/upload          | localhost
          5 | hot          | /lustre/ia2-lst1/vospace/storage/users   | localhost
          2 | local        | /lustre/ia2-lst1/users/inaf_users        | localhost
(5 rows)
```

There are two **local** storage points used for system purposes:

- the storage point with storage_id = 1 is used to store files uploaded through the web interface
- the storage point with storage_id = 2 is a special case, that is the base path of the user folders used to recall/copy huge amounts of data (scratch area).

Hot and **cold** storage points currently in use are the following:

- the storage point with storage_id = 4 is the Spectrum Scale (GPFS) mount point
- the storage point with storage_id = 5 is a Lustre mount point dedicated to the VOSpace.

Both local and hot storage points rely on the Lustre file system[1]. Data stored into a local storage point can be downloaded synchronously. Data stored into hot or cold storage points must be retrieved asynchronously. More precisely, the VOSpace creates symbolic links to make data available from hot storage points. In this way there is no need to perform a copy. Data stored into cold storage points (e.g. the tape library), instead, must be recalled.

The VOSpace database structure can be found in the `vospace-file-catalog` GitLab repository[2].

2.4 Transfer node

Name	tn-bastion.ia2
CPU(s)	4
RAM	4 GB
VIRTUAL DISKS	3
/boot (1 GB) / (10 GB) swap (4 GB)	
OS	CentOS 7

The `tn-bastion.ia2` is a general purpose transfer node. This VM allows INAF users to login via SSH with the INAF credentials, using SSSD (System Security Services Daemon)[3]. SSSD is configured to use the INAF LDAP.

2.4.1 How to create a user account

User directories are placed in `/lustre/ia2-lst1/users/inaf_users/` or simply `/inaf_users/` (symlink to the former).
To create a user account, we first need to add the corresponding username in `/etc/sssd/sssd.conf` under the **[domain/default]** section:

```
[domain/default]
...
simple_allow_users = name1.surname1, name2.surname2, ...
...
```

Then, we need to restart the SSSD service with:

```
[root@tn-bastion ~]# systemctl restart sssd
```

Finally, we need to create the user directory and provide the right permissions:

```
[root@tn-bastion ~]# cd /inaf_users/
[root@tn-bastion inaf_users]# mkdir name.surname
[root@tn-bastion inaf_users]# chown name.surname:ldap_users name.surname
[root@tn-bastion inaf_users]# chmod 700 name.surname
```

Now the user should be able to login with the INAF credentials:

```
ssh name.surname@tn-bastion.ia2
```

3 Services

The VOSpace application is split in four services:

- VOSpace UI[4]
- VOSpace REST[5]
- VOSpace File Service[6]
- VOSpace Transfer Service[7]

Here below we provide a short summary on how to manage these services.

3.1 VOSpace UI

The VOSpace UI is a web interface allowing the users to operate on the VOSpace upon authentication.
Configuration files, binary executable and logs can be found in `/home/vospace/vospace-ui`:

- `auth.properties`: RAP and GMS settings
- `vospace-ui.conf`: global configuration file
- `vospace-ui.jar`: the VOSpace UI binary
- `vospace-ui.log`: the VOSpace UI log file.

The service can be managed through a systemd unit located in `/etc/systemd/system/vospace-ui.service`:

```
[root@vospace ~]# systemctl (start|stop|restart|status) vospace-ui
```

3.2 VOSpace REST

The VOSpace REST Service provides a set of REST API endpoints for the operations performed on the VOSpace.
Configuration files, binary executable and logs can be found in `/home/vospace/vospace-ui`:

- `auth.properties`: RAP and GMS settings
- `vospace-rest.conf`: global configuration file
- `vospace-rest.jar`: the VOSpace REST binary
- `vospace-rest.log`: the VOSpace REST log file.

The service can be managed through a systemd unit located in `/etc/systemd/system/vospace-rest.service`:

```
[root@vospace ~]# systemctl (start|stop|restart|status) vospace-rest
```

3.3 VOSpace File Service

The VOSpace File Service provides functionalities to upload and download files.
Configuration files, binary executable and logs can be found in `/home/vospace/vospace-file-service`:

- `vospace-file-service.conf`: global configuration file
- `vospace-file-service.jar`: the VOSpace File Service binary
- `vospace-file-service.log`: the VOSpace File Service log file.

The service is managed through a systemd unit (`/etc/systemd/system/vospace-file-service.service`):

```
[root@vospace-tn ~]# systemctl (start|stop|restart|status) vospace-file-service
```

3.4 VOSpace Transfer Service

The VOSpace Transfer Service manages asynchronous data transfers between the transfer node and a given storage point (hot or cold).
This service, unlike the other services, is written in Python and runs within a virtual environment.
All the scripts have been placed in `/home/vospace/.venv/vospace-backend/vospace-transfer-service`.
The configuration files are:

- `/etc/vos_ts/vos_ts.conf`: global configuration file for the VOSpace Transfer Service backend
- `/etc/vos_cli/vos_cli.conf`: global configuration file for admin CLI commands (vos_data, vos_group, vos_user, vos_import, vos_job, vos_storage)

The log files are:

- `/var/log/vos_ts/vos_ts.log`: main log file
- `/var/log/vos_ts/error.log`: secondary and less important log file which keeps track of errors during the files/dirs cleanup phase (`file_cleaner.py` is periodically launched by a cron job; use `crontab -e` command as root to see the cron job configuration).

The service is managed through a systemd unit (`/etc/systemd/system/vospace-transfer-service.service`):

```
[root@vospace-tn ~]# systemctl (start|stop|restart|status) vospace-transfer-service
```

3.4.1 Redis

The VOSpace Transfer Service uses Redis[8] as local cache system.
By default, AOF + RDB persistence[9] is enabled. The RDB dump and the AOF file can be found under `/var/lib/redis/`.
Redis is used to store job queues for asynchronous jobs, but it is also used as broker for communication between the VOSpace REST Service and the VOSpace Transfer Service and, obviously also between the admin CLI tools of the VOSpace Transfer Service and the VOSpace Transfer Service itself.
The Redis configuration file is:

- `/etc/redis/redis.conf` or simply `/etc/redis.conf` (symlink to the former)

The service can be managed through a systemd unit:

```
[root@vospace-tn ~]# systemctl (start|stop|restart|status) redis
```

4 Lustre

Mounting a Lustre file system requires to have installed the Lustre client software[10].

4.1 Lustre client software installation

Here below is shown the command sequence to install the Lustre client on a CentOS 7 GNU/Linux distribution. First of all, we need to install the EPEL repository and the kernel packages.

```
yum install \
epel-release
kernel \
kernel-devel \
kernel-headers \
kernel-abi-whitelists \
kernel-tools \
kernel-tools-libs \
kernel-tools-libs-devel
```

Now, we need to create the YUM repositories:

```
cat >/etc/yum.repos.d/lustre.repo <<\__EOF
[lustre-server]
name=lustre-server
baseurl=https://downloads.whamcloud.com/public/lustre/latest-release/el7/server/
gpgcheck=0

[lustre-client]
name=lustre-client
baseurl=https://downloads.whamcloud.com/public/lustre/latest-release/el7/client
gpgcheck=0

[e2fsprogs-wc]
name=e2fsprogs-w
baseurl=https://downloads.whamcloud.com/public/e2fsprogs/latest/el7
gpgcheck=0
__EOF
```

We reboot the VM and then we install the Lustre client user-space tools and DKMS kernel module package:

```
yum --nogpgcheck --enablerepo=lustre-client install \
lustre-client-dkms \
lustre-client
```

We configure Lustre LNet device:

```
#LNet
cat > /etc/modprobe.d/lustre.conf <<EOF
options lnet networks="tcp1(ens224)"
EOF
```

We load the Lustre kernel modules:

```
modprobe -v lustre
```

Finally, we perform the mount operation:

```
mkdir -p /lustre/ia2-lst1
mount -t lustre 192.168.0.47@tcp1:192.168.0.48@tcp1:/ia2-lst1 /lustre/ia2-lst1
```

4.2 Mount points

On `vospace-tn.ia2` and `tn-bastion.ia2` VMs there are two Lustre mount points as shown in `/etc/fstab`:

```
#LUSTRE
192.168.0.47@tcp1:192.168.0.48@tcp1:/ia2-lst1/users /lustre/ia2-lst1/users lustre  defaults,
    ↪ _netdev        0 0
192.168.0.47@tcp1:192.168.0.48@tcp1:/ia2-lst1/vospace/storage /lustre/ia2-lst1/vospace/storage
    ↪   lustre  defaults,_netdev  0 0
```

Alternatively, there is the possibility to mount them manually (as root):

```
mount -t lustre -o _netdev 192.168.0.47@tcp1:192.168.0.48@tcp1:/ia2-lst1/users /lustre/ia2-
    ↪ lst1/users
mount -t lustre -o _netdev 192.168.0.47@tcp1:192.168.0.48@tcp1:/ia2-lst1/vospace/storage /
    ↪ lustre/ia2-lst1/vospace/storage
```

Notice the presence of the `_netdev` option, which indicates to the operating system that the filesystem has to be mounted after the network is available.

5 IBM Spectrum Scale (formerly GPFS)

Mounting a Spectrum Scale (GPFS) file system requires to have installed the Spectrum Scale client software[11]. More details on IBM Spectrum Scale administration commands can be found on the IBM Spectrum Scale online documentation[12].

5.1 Spectrum Scale client software installation

In the following sections we will see all the necessary steps to install the IBM Spectrum Scale client and add our node to the GPFS cluster.

5.1.1 Client configuration

First of all we need to ensure that SELinux is disabled on our VM. Then we have to setup an additional network interface with an internal network IP address.
We need to verify that in `/etc/hosts` of our node there are all the internal network addresses of all the nodes of the GPFS cluster (every node must be able to see every other node).

Now, we have to create a SSH key pair:

```
ssh-keygen -m PEM
```

Then, we have to copy the public key on all the GPFS cluster nodes:

```
ssh-copy-id tape-fe-priv-01
ssh-copy-id spectrum-client-03
ssh-copy-id ts-au-test.storage.lan
ssh-copy-id ari-l.storage.lan
ssh-copy-id test01.storage.lan
ssh-copy-id staging.storage.lan
```

To be more precise, we must copy the public key from all nodes to all nodes, including the node itself.

Now, we update the operating system packages:

```
yum update
```

If kernel packages were involved in the update, we must reboot the machine.
We need to install the following dependencies:

```
yum install kernel-headers kernel-devel m4 make cpp gcc gcc-c++ net-tools ksh libaio
```

Then, we create a new folder called IBM, we enter it and we download the Spectrum Scale client installer:

```
mkdir IBM && cd IBM
```

We launch the installer (this will simply unpack a bunch of RPMs):

```
chmod +x Spectrum_Scale_Data_Management-5.0.5.5-x86_64-Linux-install
./Spectrum_Scale_Data_Management-5.0.5.5-x86_64-Linux-install --silent
```

At this point we can install the GPFS packages:

```
cd /usr/lpp/mmfs/5.0.5.5/gpfs_rpms/ && \
rpm -Uvh \
gpfs.base-5.0.5-10.x86_64.rpm \
gpfs.docs-5.0.5-10.noarch.rpm \
gpfs.gpl-5.0.5-10.noarch.rpm \
gpfs.msg.en_US-5.0.5-10.noarch.rpm \
gpfs.java-5.0.5-10.x86_64.rpm \
gpfs.license.dm-5.0.5-10.x86_64.rpm  \
gpfs.adv-5.0.5-10.x86_64.rpm \
gpfs.gskit-8.0.55-12.x86_64.rpm && \
cd /usr/lpp/mmfs/5.0.5.5/zimon_rpms/rhel7 && \
rpm -Uvh gpfs.gss.pmsensors-5.0.5-5.el7.x86_64.rpm
```

Don't forget to disable the firewall (the simple way):

```
systemctl stop firewalld
systemctl disable firewalld
```

In case we want to keep firewalld active, we have to configure it properly. Some hints are provided by the IBM online documentation: Securing the IBM Spectrum Scale system using firewall[13]. Moreover, we have to add the following line at the end of our internal network interface configuration file:

```
ZONE=trusted
```

Now, we check if the evnironment variable `PATH` includes the GPFS command directory (`/usr/lpp/mmfs/bin/`):

```
echo $PATH
```

If not, we need to add it manually with:

```
echo 'export PATH=$PATH:/usr/lpp/mmfs/bin/' >> ~/.bashrc
```

Finally, we need to build the GPFS compatibility layer:

```
mmbuildgpl
```

5.1.2 Server configuration

First of all, we add our client node to the GPFS cluster:

```
mmaddnode -N vospace-tn.storage.lan
```

Then, we must accept the license:

```
mmchlicense client --accept -N vospace-tn.storage.lan
```

We start the GPFS daemon on our client node:

```
mmstartup -N vospace-tn.storage.lan
```

Finally, we designate our client node as a perfmon[1] node:

```
mmchnode --perfmon -N vospace-tn.storage.lan
```

5.2 Mount points

In the previous section we have talked about the Spectrum Scale client installation on the `vospace-tn.ia2` VM. This is a preparatory step in order to be able to mount the Spectrum Scale file system.
Here below are shown the mount points in `/etc/fstab`:

```
#GPFS
lv_archive_01         /ia2_tape_archive_01 gpfs       rw,mtime,relatime,dev=lv_archive_01,
    ↪ noauto 0 0
lv_generic_rw_01      /ia2_tape_generic_rw_01 gpfs       rw,mtime,relatime,dev=lv_generic_rw_01
    ↪ ,noauto 0 0
lv_stb_01             /ia2_tape_stb_01     gpfs       rw,mtime,relatime,dev=lv_stb_01,noauto 0
    ↪ 0
```

5.3 Nodes startup

If the VM `vospace-tn.ia2` goes down or, for some reason, you need to reboot it, you will need to restart the GPFS daemon on this VM from the tape library frontend (`tape-fe.ia2`) using the `mmstartup -N vospace-tn` command:

```
[root@tape-fe ~]# mmlscluster

GPFS cluster information
========================
GPFS cluster name:         tape-ia2.tape-fe-priv-01
GPFS cluster id:           7885582776184916946
GPFS UID domain:           inaf.it
Remote shell command:      /bin/ssh
Remote file copy command:  /bin/scp
Repository type:           CCR

 Node  Daemon node name         IP address     Admin node name           Designation
-------------------------------------------------------------------------------------
   1   tape-fe-priv-01          XXX.YYY.Z.KK   tape-fe-priv-01           quorum-manager-perfmon
   5   spectrum-client-03       XXX.YYY.Z.KK   spectrum-client-03        perfmon
   6   ts-au-test.storage.lan   XXX.YYY.Z.KK   ts-au-test.storage.lan    perfmon
   7   ari-l.storage.lan        XXX.YYY.Z.KK   ari-l.storage.lan         perfmon
   8   test01.storage.lan       XXX.YYY.Z.KK   test01.storage.lan        perfmon
   9   staging.storage.lan      XXX.YYY.Z.KK   staging.storage.lan       perfmon
  10   vospace-tn.storage.lan   XXX.YYY.Z.KK   vospace-tn.storage.lan    perfmon

[root@tape-fe ~]# mmgetstate -L -a

 Node number  Node name        Quorum  Nodes up  Total nodes  GPFS state   Remarks
-------------------------------------------------------------------------------------
       1      tape-fe-priv-01    1        1          7       active       quorum node
       5      spectrum-client-03 1        1          7       active
       6      ts-au-test         1        1          7       active
       7      ari-l              1        1          7       active
       8      test01             0        0          7       unknown
       9      staging            1        1          7       active
      10      vospace-tn         1        1          7       active
[root@tape-fe ~]# mmstartup -N vospace-tn
```

[1] **perf**ormance **mon**itoring node

6 Extended permissions (ACLs)

ACL functionalities require the following packages to be installed on CentOS 7:

```
[root@tn-bastion ~]# yum list installed | grep acl
acl.x86_64                          2.2.51-15.el7                  @anaconda
libacl.x86_64                       2.2.51-15.el7                  @anaconda
```

The **VOSpace File Service** on the `vospace-tn.ia2` VM needs to be able to serve all the files/dirs recalled asynchronously by the user through the **VOSpace UI**. This can be done by setting the following ACL permissions for the **vospace** user (as root):

```
setfacl -Rdm u:vospace:rx /lustre/ia2-lst1/vospace/storage/users/name.surname
setfacl -Rm u:vospace:rx /lustre/ia2-lst1/vospace/storage/users/name.surname

setfacl -Rdm u:vospace:rx /lustre/ia2-lst1/users/inaf_users/name.surname
setfacl -Rm u:vospace:rx /lustre/ia2-lst1/users/inaf_users/name.surname
```

These two pair of rules recursively set the "rx" (read-only) permissions for the vospace user on the target directory:

- **R** is recursive, which means that everything under the target directory will have the rules applied to it
- **d** is default, which means that for all future items created under that directory, these rules will apply by default
- **m** is needed to add/modify rules.

Notice that the **vospace** user on the `vospace-tn.ia2` VM needs to access also the hot storage points on Lustre, because in this case the operation of recalling files/dirs consists in creating the dirs within the retrieve folder (`/inaf_users/name.surname/vospace_retrieve`) and creating the symbolic links to the files located on the hot storage point.
On the `tn-bastion.ia2` we need to setup the following ACL permissions in order to allow each user to read/copy the files recalled asynchronously:

```
setfacl -Rdm u:name.surname:rx /lustre/ia2-lst1/vospace/storage/users/name.surname
setfacl -Rm u:name.surname:rx /lustre/ia2-lst1/vospace/storage/users/name.surname
```

7 Installation requirements

Both `vospace.ia2` and `vospace-tn.ia2` are CentOS 7 VMs and on both are installed the following Java packages:

```
java-latest-openjdk.x86_64          1:17.0.1.0.12-1.rolling.el7    @epel
java-latest-openjdk-headless.x86_64
```

For now we stick to OpenJDK 17 by excluding from updates these packages in `/etc/yum.repos.d/epel.repo`. Updating the `vospace.ia2` VM should be straightforward. Conversely, updating the `vospace-tn.ia2` VM requires a bit more attention, especially when kernel updates are involved. Indeed, a kernel update affects the Lustre and Spectrum Scale kernel modules. In case of kernel upgrade it will be necessary to verify that the version of the new kernel is supported by both the Spectrum Scale and Lustre clients. DKMS should automatically rebuild the Lustre client kernel modules, while the IBM Spectrum Scale client must be reinstalled.

8 Service deployment and installation notes

Below, we report all the necessary steps to deploy the VOSpace in production and on a staging area for testing.

8.1 Production deployment

The VOSpace services are automatically deployed on the respective machines using the GitLab CI/CD and Ansible[14].
All the scripts, config files and pipelines are available on the `vospace-ansible` GitLab repository[15].
In the `vospace-ansible` repository we can find two different configurations:

- production configuration: is stored under `ansible/distributed` (services deployed on both `vospace-tn.ia2` and `vospace.ia2` VMs)
- test configuration: is stored under `ansible/standalone` (all services deployed on a single test VM).

Production deployment jobs can be launched in the following order:

1. deploy_vospace_tn
2. deploy_vospace_ia2.

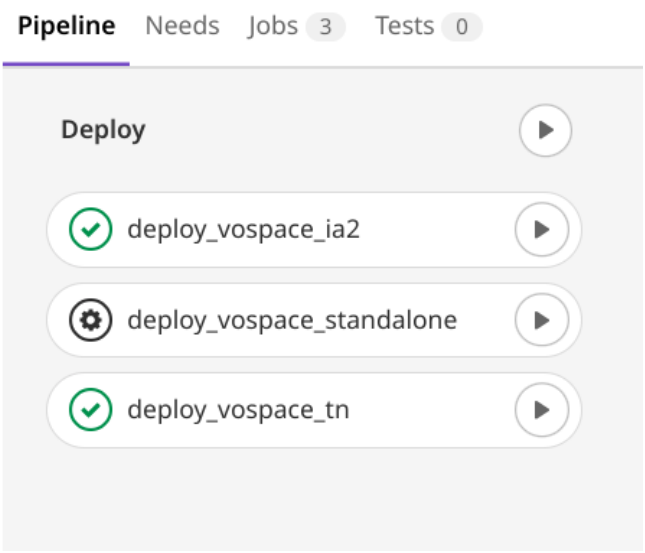

Figure 2: VOSpace GitLab CI/CD deployment pipeline for production

The deploy_vospace_standalone pipeline was used for testing a standalone installation of the VOSpace on a single virtual machine and must be ignored during the production deployment.

8.2 Deploying the VOSpace on the IA2 staging area

The `staging.ia2` VM provides a dockerized version of the VOSpace for testing purposes. The VOSpace services can be found here: `http://staging.ia2.inaf.it/`. We can use the VOSpace from the web interface, but we can also test the CLI VOSpace administration commands by accessing the appropriate docker containers through Guacamole[16].

The VOSpace can be deployed on the staging VM by using the GitLab pipeline of the `vospace-staging` GitLab repository[17]:

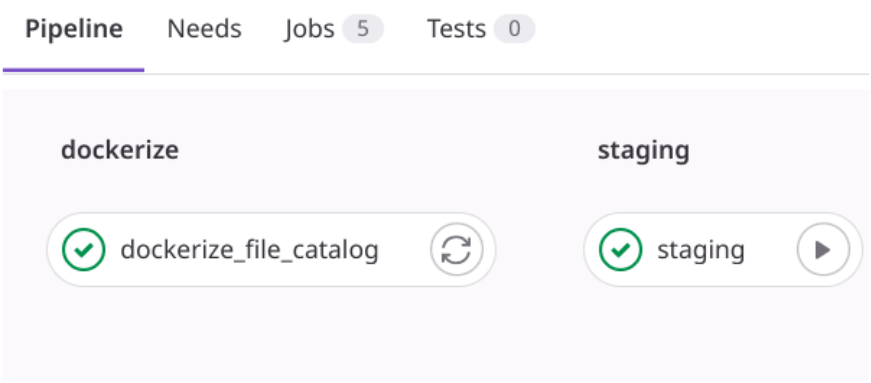

Figure 3: VOSpace GitLab CI/CD deployment pipeline for staging.ia2 VM

The jobs are launched in the following order:

1. dockerize_file_catalog
2. staging.

We can have full access to all the VOSpace containers by using SSH to login on the staging machine. For example, we can access the container running the VOSpace database:

```
[cristiano@fedora ~]$ ssh root@staging.ia2
root@staging.ia2's password:
Last login: Wed Dec 28 13:32:03 2022 from urban-vm.ia2
[root@staging ~]# su - gitlab-runner
Ultimo accesso: mar dic 20 09.09.21 CET 2022
[gitlab-runner@staging ~]$ docker exec -it file_catalog /bin/bash
root@d27aae73c84b:/# su - postgres
postgres@d27aae73c84b:~$ psql -U postgres
psql (12.5 (Debian 12.5-1.pgdg100+1))
Type "help" for help.

postgres=# \c vospace_testdb
You are now connected to database "vospace_testdb" as user "postgres".
vospace_testdb=#
```

Important: be sure to configure the VOSpace testing installation so that it can write data only on tapes assigned to the `pl_generic_rw_01` pool of the tape library. This can be done by setting `tape_pool = pl_generic_rw_01` in `conf/vos_ts.conf` on the `vospace-staging` GitLab repository, before to deploy the VOSpace on the staging machine.

9 VOSpace administration commands

Here below we briefly describe the VOSpace CLI administration commands.

9.1 vos_data

This command launches a job to automatically store data provided by the user on a given storage point (hot or cold) and import the related VOSpace nodes in the database.
Although this tool provides a good degree of automation on the operations preliminary to database import (basic data structure checks, .tar creation, MD5 checksums calculation, data copying), in production we usually

prefer to operate on data in a script aided semi-manual mode and then use the `vos_import` command. For this kind of operations, some useful scripts are available within the `ict-scripts`[18] and `Checksum utils`[19] GitLab repositories.

```
NAME
       vos_data

SYNOPSYS
       vos_data COMMAND USERNAME

DESCRIPTION
       The purpose of this client application is to notify to the
       VOSpace backend that data is ready to be saved somewhere.

       The client accepts only one (mandatory) command at a time.
       A list of supported commands is shown here below:

       cstore
              performs a 'cold storage' request, data will be saved on tape

       hstore
              performs a 'hot storage' request, data will be saved to disk

       The client also needs to know the username associated to a storage request process.
       The username must be the same used for accessing the transfer node.

EXAMPLES
       1) Perform a hot storage request for the data contained in the 'store path' associated
       to the user 'john.smith':

       # vos_data hstore john.smith

       2) Perform a cold storage request for the data contained in the 'store path' associated
       to the user 'john.smith':

       # vos_data cstore john.smith

       The 'store path' parameter is defined in the 'storage section' of the transfer service
       main configuration file: '/etc/vos_ts/vos_ts.conf'.
```

9.2 vos_group

Allows to add/remove groups to/from `group_read` and/or `group_write` properties of a VOSpace node and its child nodes (if any). It also allows to list all groups associated to a node.

```
NAME
       vos_group

SYNOPSYS
       vos_group GROUP_TYPE METHOD [GROUP_NAME] VOSPACE_PATH

DESCRIPTION
       This tool allows to modify 'group_read' and 'group_write'
       parameters of a VOSpace node.

       Four parameters are required:

       GROUP_TYPE:
           specifies the group type: allowed values are 'read' or 'write'

       METHOD:
           there are three supported methods:
               1. 'add': adds a group to a node
               2. 'del': removes a group from a node
               3. 'list': lists all groups associated to a node
```

```
       GROUP_NAME:
           represents a group of users or a single user.
           In the first case, just specify the group name.
           In the second case, the group name syntax is:

               people.name\\.surname

       VOSPACE_PATH:
           represents the node VOSpace path.

       EXAMPLES
           1) Add 'jane.lee' to 'group_read' for the VOSpace node '/john.smith/test/foo' and
               ↪ any child nodes:

           # vos_group read add people.jane\\.lee /john.smith/test/foo

           2) Add 'my_group' to 'group_write' for the VOSpace node '/john.smith/test/foo' and
               ↪ any child nodes:

           # vos_group write add my_group /john.smith/test/foo

           3) Remove 'my_group' from 'group_write' for the VOSpace node '/john.smith/test/foo'
               ↪  and any child nodes:

           # vos_group write del my_group /john.smith/test/foo

           4) List all groups in 'group_read' for the VOSpace node '/john.smith/test/foo':

           # vos_group read list /john.smith/test/foo
```

9.3 vos_import

Imports VOSpace nodes in the database (file catalog) for data already stored on a given storage point.

```
NAME
       vos_import

SYNOPSYS
       vos_import DIR_PATH USERNAME

DESCRIPTION
       This tool recursively imports nodes on the VOSpace file catalog.

       Two parameters are required:

       DIR_PATH:
           the physical absolute path of a directory located in the first
           level of the user directory for a given mount point

       USERNAME:
           the username used for accessing the transfer node

       EXAMPLE
           1) The following command will import recursively all the nodes contained
           in 'mydir' on the VOSpace for the 'jsmith' user:

           # vos_import /mnt/storage/users/jsmith/mydir jsmith

           2) The following command will import recursively all the nodes contained
           in the timestamp-wrapper-dir '2021_03_03-13_46_34-vos_wrapper' on the
           VOSpace for the 'jsmith' user:

           # vos_import /mnt/storage/users/jsmith/2021_03_03-13_46_34-vos_wrapper jsmith
```

9.3.1 Importing ARI-L data

ARI-L[2] is a development project providing data reduction pipelines for the ALMA Science Archive[3]. The ARI-L data is organized in one root folder (`ari-l`) with a huge number of subfolders at the first level, called MOUs. These MOUs are saved on tape in several tranches for long term preservation, then imported into the VOSpace and made publicly visible on the VOSpace UI. Each data tranche may comprise several hundred MOUs and its ingestion is made one-shot. All these peculiarities led us to consider a specific approach for data ingestion and import, potentially extendable to similar cases.
Since the `vos_import` command requires as argument a path being in the first level of the project directory, if we need to import a tranche of data, we find ourselves having to launch one import job for each one of these first level subfolders being part of the tranche.
In this specific case, we need to access the `vospace-tn.ia2` VM and setup in `/etc/vos_ts/vos_ts.conf` the **max_ready_jobs** parameter to a value which must be greater than the number of MOUs to import. We also have to temporarily disable email notifications in order to avoid a huge amount of email messages to be sent (one for each job). This can be done by setting the **enable_notifications** parameter to `false` in `/etc/vos_ts/vos_ts.conf` under the [**mail**] section. Then, we must restart the VOSpace Transfer Service.
After copying the file containing the list of MOUs to import (e.g. `ari-l-tranche.txt`) and the import script to `/home/vospace` of the vospace-tn.ia2 VM, we can launch the import batch:

```
[root@vospace-tn ~]# su - vospace
(vospace) [vospace@vospace-tn ~]$ chmod +x import_from_dir_list
(vospace) [vospace@vospace-tn ~]$ screen
(vospace) [vospace@vospace-tn ~]$ ./import_from_dir_list
```

The `import_from_dir_list` file is a Bash script containing the following instructions:

```
#!/bin/bash

set -e

while read i;
  do
    mou=$(echo $i | cut -d '/' -f 2)
    vos_import /ia2_tape_stb_01/users/ari-l/${mou} ari-l
    sleep 0.2
  done < ari-l-tranche.txt
```

The `ari-l-tranche.txt` file shall contain the names of the folders to import, one per row. Here below is shown a possible content sample:

```
./2013.1.01046.S_uid___A001_X145_X249
./2013.1.00812.S_uid___A001_X146_X1e
./2013.1.00014.S_uid___A001_X12b_X235
./2013.1.00220.S_uid___A001_X122_X125
./2016.1.00496.S_uid___A001_X87a_X241
./2013.1.00338.S_uid___A001_X12a_X1fd
./2013.1.00870.S_uid___A001_X12f_X6
./2013.1.00433.S_uid___A001_X13b_Xfa
./2015.1.00139.S_uid___A001_X2f7_Xb9
./2013.1.00379.S_uid___A001_X145_X18e
./2015.1.01339.S_uid___A001_X2fb_Xc15
./2013.1.00048.S_uid___A001_X137_Xd
./2013.1.00781.S_uid___A001_X144_Xeb
./2013.1.00214.S_uid___A001_X146_Xd0
./2013.1.01195.S_uid___A001_X122_X586
./2013.1.01194.S_uid___A001_X11f_X4c
./2015.1.01144.S_uid___A001_X2fe_X4cf
./2013.1.01391.S_uid___A001_X147_X27c
./2013.1.01046.S_uid___A001_X145_X24c
./2013.1.00166.S_uid___A001_X12b_X158
./2013.1.00586.S_uid___A001_X13e_X151
./2013.1.00234.S_uid___A001_X121_X1bf
```

[2] **A**dditional **R**epresentative **I**mages for **L**egacy
[3] https://almascience.eso.org/

Obviously, at the end of this task we must restore the previous configuration parameters and then restart the VOSpace Transfer Service.

9.4 vos_job

Provides information about jobs.

```
NAME
       vos_job

SYNOPSYS
       vos_job COMMAND [ARGUMENT]

DESCRIPTION
       Client tool to obtain information about jobs.

       The client accepts the following commands:

       list
           if launched without any argument it will display all the currently active jobs.
           The command can also be used with one argument, the job phase. The following
           values are allowed:
           - pending
           - queued
           - executing
           - completed
           - error

       info
           prints a JSON object containing the job info according to the UWS specification.
           A job ID is required as argument

       search
           performs a search on jobs and returns those having a match between the search
               ↪ string
           passed by the user and one of the following fields:

           'job_id', 'job_type', 'ownder_id', 'user_name'

       results
           prints a JSON object containing the job results according to the UWS specification.
           A job ID is required as argument.
```

9.5 vos_storage

Allows to add, list and remove storage points.

```
NAME
       vos_storage

SYNOPSYS
       vos_storage COMMAND

DESCRIPTION
       Client tool to manage VOSpace storage points.

       The client accepts only one (mandatory) command at a time.
       A list of supported commands is shown here below:

       add
           adds a storage point to the database

       del
           deletes a storage point from the database
```

```
       list
           prints the full storage point list


       Supported storage types are: 'hot' and 'cold'.

       Adding 'hot' or 'cold' storage points requires a base path
       to be specified (e.g. '/mnt/my_storage/users').

       All storage points require a valid hostname or IP address.
```

9.6 vos_user

Allows to add/remove users to/from the `user` table of the VOSpace database.

```
NAME
       vos_user

SYNOPSYS
       vos_user COMMAND [ARGUMENT]

DESCRIPTION
       Client tool to manage VOSpace users in the database.

       The client accepts only one (mandatory) command at a time.
       A list of supported commands is shown here below:

       add
           adds a user to the database

       del
           deletes a user from the database

       search
           performs a search on users and returns those having a match between the search
               ↪ string
           passed via command line and one of the following fields:

           'user_id', 'user_name', 'e_mail'

           Adding a user to the database requires a user ID, a username and an e-mail address.
           A valid userID is required when deleting a user from the database.

       IMPORTANT NOTES:
       the VOSpace Transfer Service automatically populates the 'users' table in the database
       by previously quering the authentication system (RAP).
       It also adds the user node in the 'node' table, if not present (e.g. '/john.smith' for
       the user 'john.smith'). You can use this client to register a user manually, as needed.
       For all other cases, please, use this client only if you need to handle situations
       involving users that for some reason are not recognized by the authentication system.
```

10 Useful SQL queries for administrators and developers

In this last section we provide a short selection of SQL queries that might be useful during some administration tasks.

1) Obtain all the node paths for a specific user:

```
[root@vospace-tn ~]# su - vospace
(vospace) [vospace@vospace-tn ~]$ vos_user search cristiano.urban

+---------+-----------------+-------------------------+
| user_id |    user_name    |         e_mail          |
```

```
+---------+----------------+-------------------------+
|  3354   | cristiano.urban | cristiano.urban@inaf.it |
+---------+----------------+-------------------------+

(vospace) [vospace@vospace-tn ~]$ ssh root@db02.ia2
[root@db02 ~]# su - postgres
-bash-4.2$ psql vospace -c "SELECT name, path, fs_path FROM node WHERE creator_id = '3354';"
```

2) Obtain the total number of completed jobs for a given user:

```
-bash-4.2$ psql vospace -c "SELECT count(*) FROM job WHERE owner_id = '3354' AND phase = '
    ↪ COMPLETED';"
```

3) Obtain the list of nodes contained in `ROOT/cristiano.urban`:

```
-bash-4.2$ psql vospace -c "SELECT name, creator_id FROM node WHERE creator_id = '3354' AND
    ↪ name <> 'cristiano.urban';"
```

4) Get the total number of nodes in the VOSpace:

```
-bash-4.2$ psql vospace -c "SELECT count(*) FROM node;"
```

5) Get the total number of nodes contained within the `ROOT/ari-l` container node:

```
-bash-4.2$ psql vospace -c "SELECT count(*) FROM node WHERE parent_path = text2ltree(
    ↪ id_from_vos_path('/ari-l')::text);"
```

6) Get node `name`, `creator_id` from `fspath` for a node created via VOSpace UI:

```
-bash-4.2$ psql vospace -c "SELECT name, creator_id FROM node WHERE fs_path ~ '20f799b8-d41f-4
    ↪ e2b-b9d5-fe98562f542f';"
```

7) Find information about a specific user node:

```
-bash-4.2$ psql vospace -c "SELECT name, content_md5 FROM node WHERE path <@ text2ltree(
    ↪ id_from_vos_path('/franco.vazza')::text) AND name = 'lista.txt';"
```

8) Find all child nodes for a given VOSpace path:

```
psql vospace -c "SELECT name, path FROM node n WHERE n.parent_path <@ (SELECT path FROM node
    ↪ WHERE node_id = id_from_vos_path('/cristiano.urban/test/test_b'));"
```

9) Update a storage point base path:

```
-bash-4.2$ psql vospace -c "UPDATE storage SET base_path = '/lustre/ia2-lst1/vospace/users'
    ↪ WHERE storage_id = 2;"
```

10) Make `ari-l` data public:

```
-bash-4.2$ psql vospace -c "UPDATE node SET is_public = 'true' WHERE path <@ text2ltree(
    ↪ id_from_vos_path('/ari-l')::text);"
```

Or, more simply:

```
-bash-4.2$ psql vospace -c "UPDATE node SET is_public = 'true' WHERE creator_id = 'ari-l';"
```

11) Delete all the user nodes, excluding the user home container node:

```
-bash-4.2$ psql vospace -c "DELETE FROM node WHERE path <@ text2ltree(id_from_vos_path('/
    ↪ cristiano.urban')::text) AND name <> 'cristiano.urban';"
```

12) Update the `creator_id` for all the nodes of a given user:

```
-bash-4.2$ psql vospace -c "UPDATE node SET creator_id = '3883' WHERE creator_id = '2060';"
```

This query comes in handy when the `user_id` field of the VOSpace `user` table is different from the `user_id` field in the `identity` table of the RAP[20] database. In fact, it may happen that user data previously imported in the database with one user id could not be visible by the user because the `user_id` changed in a second moment.

13) Update the `content_length` field of all the container nodes (this may take a while):

```
-bash-4.2$ psql vospace -c "UPDATE node n SET content_length = (SELECT sum(content_length)
    ↪ FROM node m WHERE m.parent_path <@ n.path AND type = 'data') WHERE type = 'container';"
```

This query updates the size of all the container nodes. This feature may be probably implemented in the future. We can see some statistics about performances by prepending `EXPLAIN ANALYZE` to the SQL query.

11 Final remarks and future developments

In this document we have described the INAF VOSpace architecture from the sysadmin point of view. We have seen an overview on machines configurations, service management, installation, configuration and deployment steps and some further useful tips.
As for possible future developments, we may think to explore more efficient upload/download options for large VOSpace data (e.g. SFTP[4] and GridFTP[21]).

[4] https://it.wikipedia.org/wiki/SSH_File_Transfer_Protocol

References

[1] *The Lustre file system*. URL: https://www.lustre.org/.

[2] *The VOSpace File Catalog GitLab repository*. URL: https://www.ict.inaf.it/gitlab/vospace/vospace-file-catalog.

[3] *System Security Services Daemon*. URL: https://sssd.io/.

[4] *The VOSpace UI GitLab repository*. URL: https://www.ict.inaf.it/gitlab/vospace/vospace-ui.

[5] *The VOSpace REST GitLab repository*. URL: https://www.ict.inaf.it/gitlab/vospace/vospace-rest.

[6] *The VOSpace File Service GitLab repository*. URL: https://www.ict.inaf.it/gitlab/vospace/vospace-file-service.

[7] *The VOSpace Transfer Service GitLab repository*. URL: https://www.ict.inaf.it/gitlab/vospace/vospace-transfer-service.

[8] *Redis*. URL: https://redis.io/.

[9] *Redis persistence*. URL: https://redis.io/docs/management/persistence/.

[10] *Installing the Lustre Software*. Sept. 2020. URL: https://wiki.lustre.org/Installing_the_Lustre_Software.

[11] *Manually installing the IBM Spectrum Scale software packages on Linux nodes*. July 2022. URL: https://www.ibm.com/docs/en/spectrum-scale/5.1.1?topic=isslndp-manually-installing-spectrum-scale-software-packages-linux-nodes.

[12] *IBM Spectrum Scale command reference*. Dec. 2022. URL: https://www.ibm.com/docs/en/spectrum-scale/5.1.5?topic=command-reference.

[13] *Securing the IBM Spectrum Scale system using firewall*. July 2022. URL: https://www.ibm.com/docs/en/spectrum-scale/5.1.1?topic=topics-securing-spectrum-scale-system-using-firewall.

[14] *Ansible*. URL: https://www.ansible.com/.

[15] *The vospace-ansible GitLab repository*. URL: https://www.ict.inaf.it/gitlab/vospace/vospace-ansible.

[16] *Apache Guacamole*. URL: https://guacamole.apache.org/.

[17] *The vospace-staging GitLab repository*. URL: https://www.ict.inaf.it/gitlab/vospace/vospace-staging.

[18] *The ict-scripts GitLab repository*. URL: https://www.ict.inaf.it/gitlab/ia2/ict-scripts.

[19] *The Checksum utils GitLab repository*. URL: https://www.ict.inaf.it/gitlab/ia2/checksum-utils.

[20] *Remote Authentication Portal (RAP)*. URL: https://sso.ia2.inaf.it/rap-ia2/.

[21] *GridFTP*. URL: https://gridcf.org/gct-docs/latest/gridftp/index.html.