OwnCloud Black Book Documentation

Release 1

Jens-Christian Fischer <jens-christian.fischer@switch.ch>

February 23, 2015

CONTENTS

.git@github.com:switch-ch/cloudservice-owncloud.git. OwnCloud Black Book documentation master file, created by sphinx-quickstart on Wed Mar 26 17:16:26 2014. You can adapt this file completely to your liking, but it should at least contain the root *toctree* directive.

Contents:

CHAPTER

ONE

INTRODUCTION

1.1 Licence

1.2 Goals

As we NRENs start to implement cloud services, we are all facing similar challenges and problems. In order to reduce the amount of separate work, discovery and solving of problems, the NREN community is collaborating on a set of guides for various services that cover items from the following list:

- Service Definition
- Implementation
- Operations
- Legal Aspects / SLA
- Tariff and Business Models
- Real World Experiences / Use Cases

This book specifically covers a file sync and share service based on ownCloud. However, the non-technical chapters are applicable to other sync & share services.

1.3 Audience

Project teams in NRENs that are tasked with designing, building, implementing and running a ownCloud based service for their community.

1.4 History

This book is a collaborative effort of the European NREN community, organized in TERENA and GEANT. It has started as a project of the GN3+ SA7 project.

The technical parts are based on the implemention of ownCloud at SWITCH, with the SWITCHdrive project. It can serve as one example of how one might technically build such a service.

As of October 2014 a section describing the implementation of ownCloud at GARR, with the GARRbox service, has been added.

In January 2015, CESNET has added a description of its ownCloud@CESNET service.

1.5 Authors

- Jens-Christian Fischer
- Patrik Schnellmann
- the GARRbox team
- Miroslav Bauer, Jan Hornicek and David Antos @ CESNET
- Your name here

CHAPTER

TWO

SERVICE DESCRIPTION

The success of commercial file sharing solutions like Dropbox, Google Drive and others have led many users to utilize those service for file and document sharing and collaboration.

Information that is stored on one computer can easily be synced to other computers owned by the same user, mobile devices and also shared with colleagues and collaborators.

Members of universities and institutions in general use their own devices to manage their data as well as data the is processed in the context of the institution. A survey of SWITCH showed that about 55 percent of university members who use Dropbox use it to store university data, most of the time along with private data. In the view of the management of those institutions this is problematic due to legal reasons, protection of intellectual property and sometimes even not in line with the current policies. This makes the case for local sync and share services and calls for a provider to host the service. In the higher education world, hosting of such a service is naturally a task for an NREN.

SWITCH evaluated ownCloud together with the IT services of the universities and beta test users who provided feedback. In short, the conclusion was that the key success factor for a service was the uptake by users. A solution with functionality and a user interface (usability) comparable to commercial solutions seemed to be the candidate to support that factor.

ownCloud was one of the solutions to fulfil all of the following key requirements:

- On-premise solution (Swiss data location and compliance with Swiss data protection law)
- High usability (user interface and functionality)
- Support of desktop and mobile operating systems
- Access control with AAI

CHAPTER

THREE

EXPERIENCES

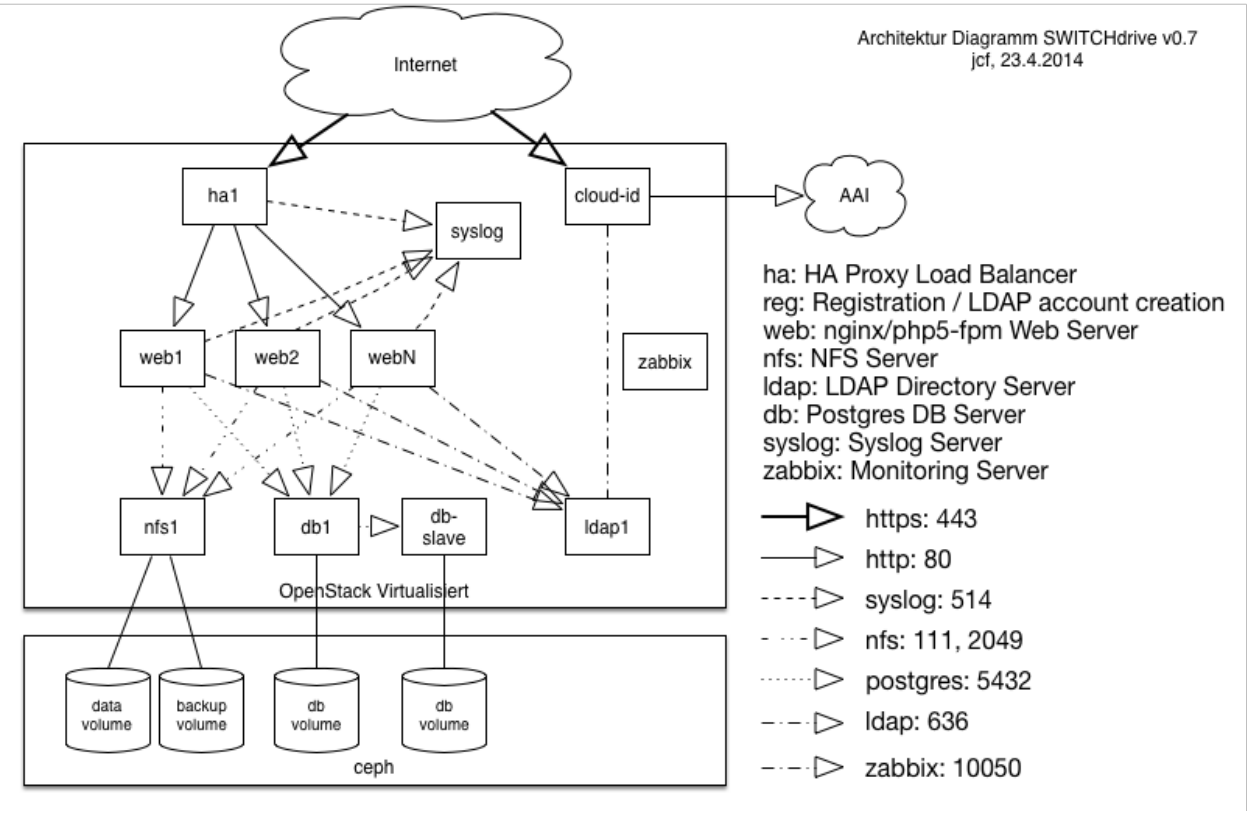
This chapter summarises some of the technical experiences we have had while deploying and running ownCloud as a shared service.

CHAPTER

FOUR

SWITCHDRIVE ARCHITECTURE

The following picture shows the current architecture diagram.



The current architecture is not HA (High Availability) enabled. As all the VMs are running on physical hosts, if one of them breaks, then potentially multiple VMs are affected at the same time. For now, I choose the path of being able to restart the VMs on another physical host in case of failure.

4.1 Common

All virtual machines are running Ubuntu 13.10 in a standard server installation. They have IPtables installed and only expose TCP port 22 plus the ports necessary for the service to the internal 10.0.20.0/24 network.

4.2 Loadbalancer

Incoming requests are handled by HAProxy. HAProxy does load balancing and SSL termination. Doing SSL termination on the load balancer allows it to inject cookies into the HTTP traffic that enables it to provide "sticky sessions" (i.e. requests from a single client always go to the same web server).

The HAProxy log files are sent to the syslog server.

HAProxy runs completely in memory and doesn't touch the disks while running. It uses a single-threaded evented execution model and therefore is quite happy on a 2 VCPU machine. To increase performance, it helps to pin the virtual CPUs to physical CPUs on the host running the VM.

Open Ports:

- 22
- 80
- 443

4.3 Web/Application servers

The web/application servers are handled by nginx and PHP5 FPM.

nginx handles all incoming HTTP requests from the load balancer, serves static files (css,js as well as the actual files stored in ownCloud) and passes all requests to ownCloud to the PHP5-FPM task.

nginx is configured with 32 worker processes (http://wiki.nginx.org/CoreModule#worker_processes) in order to maximize the amount of files that can be sent when waiting for disk io.

PHP-FPM spawns a number of PHP worker processes on startup and manages them automatically (spawning more if the load rises and killing them again if load decreases). The goal is to have a PHP process ready whenever a request comes in. (The process manager will always keep 8 PHP processes running and spawn up to 30 processes when under load)

Each web server mounts the ownCloud data directory as `/mnt/data` from the NFS server.

The ownCloud logfiles are sent to the syslog server

The web servers are 8 VCPU, 16GB RAM virtual machines. The amount of RAM is so high because ownCloud allows large files to be uploaded through the web UI. The maximal file size is 4GB, but during the upload, the file is in memory. Lower amounts of RAM could lead to situations where the memory on the server is being exhausted.

Open Ports:

- 22
- 80

4.4 Database Server

Instead of MySQL we have opted to use PostgreSQL (currently version 9.3) to store the database.

The database is backed up every 6 hours via a crontab entry. Future improvments will include DB snapshot and WAL archiving.

ownCloud is read heavy (factor 1000:1) so a possible performance option will be to have Postgres read slaves and a Postgres aware load balancer PgPool II to separate write and read requests.

The VM is configured with 32 GB of RAM and 8 VCPUs. The database currently is tuned for those settings, so resizing the VM needs to take that into account (both postgres.conf and the virtual memory settings of the server need to be adapted)

Open Ports:

- 22
- 5432

4.5 Database Slave Server

We are using Postgres streaming replication to keep a hot standby database server. The slave receives all changes from the master server and is able to

- take over the role of the master in case of emergency (procedure needs to be described)
- act as a read slave to distribute read load

Open Ports:

- 22
- 5432

4.6 NFS Server

OwnCloud needs access to a file system where it stores the users files. Because we have multiple web servers and each server needs access to the same file system, we have a 20 TB volume that is mounted in a VM and exposed with an NFS server.

The 20TB volume is a RBD volume backed by Ceph. It is thin provisioned and can be resized to accomodate for future growth.

An additional 20TB volume is used for generational file system backups with RSnapshot. This is mainly a safeguard for the case when ownCloud should lose data.

There hasn't been much tuning of the NFS server yet (save for mounting the volumes asynchronously and enabling RBD caching for the VM). The clients use 1MB large read and write buffers when mounting the NFS server. Benchmarks have shown that the NFS server can write around 120MB/s, which should be enough for now.

On initial syncs or rapid reads of ownCloud data, we notice huge spikes of CPU activity. Should that turn out to be a problem, we can potentially throttle the IOPS that the VM is allowed to perform - that could make an improvement in CPU load (lower overall IOPS but sustained performance even under load)

Open Ports:

- 111 (tcp/udp)
- 2049 (tcp/udp)

4.7 LDAP Server

The OpenLDAP server provides the authentication service for the ownCloud installation (and also for other SWITCH cloud based services). It is a single LDAP server with a directory structure adapted to the needs of the different cloud projects.

The database of the LDAP server is backed up daily in .ldif format (into `/var/backup/slapd`

Open Ports:

- 22
- 636

4.8 Syslog Server

The rsyslog server collects the logfiles from ownCloud (the application) and the access logs from the haproxy server.

Those files are stored on a separate 100GB volume, mounted at `/var/log`

Open Ports:

- 514

4.9 CloudID Server

The cloud id server is used to bridge between AAI and the LDAP server. External users can login with AAI and are able to create a new account (that will be commissioned on the LDAP server) or to reset their password.

The server runs a Ruby on Rails application developed by SWITCH's Interaction Enabling team. It uses Apache and mod_shib, and xxx as its database.

Open Ports:

- 22
- 80
- 443

4.10 Monitoring Server

The monitoring server runs Zabbix and collects statistics from the different virtual machines and provides statistics and graphs.

It runs Apache with PHP and Postgres. While the initial configuration of the Zabbix server is done automatically, the actual configuration of monitored servers is done manually.

Open Ports:

- 22
- 443
- 10050
- 10051

4.11 Thoughts on High Availability

The current setup is not HA (high-availability) one. While it certainly would be possible to build a complete HA setup, we have decided against this for a number of reasons:

- We don't expect the virtual machines to fail. If hardware fails, it will be the physical hypervisors. In the current (smallish) deployment, there are not enough machines to make the failure of just one not taking down multiple of the ownCloud VMs.
- IP failover in the OpenStack environment is a bit complicated (we can use the `nova` command line API to switch a floating IP to another VM, but this is not well integrated with the common HA solutions (`heartbeat/corosync` or `keepalived`

In case of failure of a physical host or a VM, we are prepared to experience some downtime. In case of a failed host, the dead VMs can be restarted on another physical host or rebuilt using the ansible scripts within a few minutes.

CHAPTER

FIVE

SWITCHDRIVE INSTALLATION

5.1 Prerequisites

5.1.1 Software

You will need to have Git and Ansible 1.4/1.5 installed on your local machine. Ansible installation takes you through the steps to get Ansible up and running on your local machine.

5.1.2 SSH configuration

If you use an OpenStack-based cloud environment, the various virtual machines won't have any public IP addresses (except for the load balancer). In order to get SSH access to the VMs, you will need to do a bit of configuration on a jumposthost SWITCHcloud to host the virtual machines, you will need to add the following to your `~/.ssh/config` to get direct SSH access to the 10.0.20.0/24 net (or whatever it is in your configuration) the VMs run on:

```
Host *.example.ch
  ForwardAgent yes


Host 10.0.20.*
  LogLevel error
  ForwardAgent no
  ForwardX11 no
  CheckHostIP no
  StrictHostKeyChecking no
  UserKnownHostsFile=/dev/null
  ProxyCommand ssh jump.cloud.example.com nc %h %p
  controlmaster auto
  ControlPath /tmp/ssh_mux_%h_%p_%r
```

Also, you will need to have a user account on `jump.cloud.example.com` (or whatever your host is called).

5.1.3 Clone Ansible project

Clone the ansible project on your local machine:

```
$ cd work
$ git clone github.com:/to/be/defined.git
```

5.1.4 Anatomy of the Ansible project

The project is set up with several folders and files:

- backups - will hold the downloaded database backup files (empty at start)
- group_vars - variables for the different server groups (mostly open tcp/udp ports)
- library - additional ansible tasks
- roles - the meat of the configuration. Contains the setup instructions for the different roles
- ssl - storage for SSL certificates and keys (not checked in)

Files:

- staging - IP addresses and config variables for **staging** environment
- production - IP addresses and config variables for **production** environment
- db,dev,lb,ldap,nfs,syslog,web}servers.yml - specification of the various server classes
- various commands for [[switchdrive:commonoperations|operating]] the cluster

5.1.5 Generate VMs

Create the necessary number of virtual machines for an installation of SWITCHdrive. You will need machines for the following roles:

- Loadbalancer (2 VCPU, 4 GB RAM, 20 GB Disk, Public IP Address)
- Webserver (8 VCPU, 16 GB RAM, 20 GB Disk)
- DB server (8 VCPU, 32 GB RAM, 40 GB Disk)
- DB slave server (8 VCPU, 32 GB RAM, 40 GB Disk)
- LDAP server (2 VCPU, 4 GB RAM, 20 GB Disk)
- NFS server (2 VCPU, 4 GB RAM, 20 GB Disk)
- Cloud ID server (1 VCPU, 2 GB RAM, 20 GB Disk)
- Monitoring server (2 VCPU, 4 GB RAM, 20 GB Disk, Public IP Address)

Boot each VM and run through installation of OS. Make a note of the internal IP addresses of the VMs.

Create the necessary volumes:

- data volume (/dev/vdb on NFS server)
- backup volume (/dev/vdc on NFS server)
- log volume (/dev/vdb on syslog server)
- db volume (/dev/vdb on db server)

5.1.6 Configuration of Environment

Create an **environment** file in the ansible directory. Here is an example of the **staging** file:

```
[db]
10.0.20.45 ansible_ssh_user=ubuntu

[nfs]
10.0.20.61 ansible_ssh_user=ubuntu

[web]
10.0.20.3 ansible_ssh_user=ubuntu

[lb]
10.0.20.63 ansible_ssh_user=ubuntu

[ldap]
10.0.20.62 ansible_ssh_user=ubuntu

[monitoring]
10.0.20.64 ansible_ssh_user=ubuntu

[syslog]
10.0.20.67 ansible_ssh_user=ubuntu

[dev]

[cmd]
localhost ansible_connection=local

[staging:children]
db
nfs
web
lb
ldap
monitoring
syslog
cmd

[staging:vars]
service_name=drive-stage.switch.ch
ldap_ip=10.0.20.62
ldap_host=stage-ldap
ldap_password=ldap_secret_password
nfs_ip=10.0.20.61
db_ip=10.0.20.45
syslog_ip=10.0.20.67
admin_pass=owncloud_admin_password
OWNCLOUD_VERSION='6.0.2'
```

The first part tells ansible which virtual machine is in which group (the groups are `[db]`, `[web]` etc. Note that a group can have more than one group (however, the setup only works for the web group having multiple servers. All other groups should only have one server)

The `[staging:children]` group collects all the servers into one group, so that the next section `[staging:vars]`, the variables, are visible to all configured servers.

5.1.7 Initial Installation on VMs

You can either install VMs from stock Ubuntu 13.10 images or create a volume first, and then boot the VM from the volume.

The current setup is a mix of image and volume based servers. The original thought was to make the servers boot from volume in order not to lose data due to ephemeral disks being deleted. However, I am now (March 2014) in the process of switching the servers to being image based with the databases/persistent data on volumes. This makes for much easier recreation of VMs should they fail.

Current status:

VM	Status	Notes
ldap	Volume	LDAP db needs to be persistent
nfs	Volume	Can be moved to image as no data is persistent
lb	Volume	ditto
web	Volume	ditto
db	Image	database stored on separate volume
syslog	Image	log data stored on separate volume
zabbix	Volume	database needs to be moved to volume

The following volumes need to be created:

- owncloud data (20TB) - nfs /dev/vdb
- owncloud backups (20TB) - nfs /dev/vdc
- sylog logs (100GB) - syslog /dev/vdb
- db data (100GB) - db /dev/vdb

and attached to the correct servers.

The file systems on the syslog and db servers are created automatically by the ansible playbook. In general we create the filesystem directly on the disk, without partitioning it. This allows the volume to be resized without resizing a partition on it, which makes the process simpler.

Due to historical reasons, the owncloud data lies on a partition (/dev/vdb1) so you need to manually create that partition and format it:

```
# on the NFS server
parted /dev/vdb    # and then create a logical partition
mkfs.xfs /dev/vdb1
mkfs.xfs /dev/vdc
```

5.1.8 Manual configurations

You can (and should) edit the variables in the environment file (`staging`, `production`).::

```
[staging:vars]
# the dns entry name of the service (maps to the public IP of the loadbalancer)
service_name=drive-stage.example.com

# the internal IP address of the LDAP Server (same as the one above)
ldap_ip=10.0.20.62

# The host name of the LDAP server
ldap_host=stage-ldap

# The password that the cn=admin account has on the LDAP server
```

```
ldap_password=secret_ldap_password

# The internal IP address of the NFS server
nfs_ip=10.0.20.61

# The internal IP address of the database server
db_ip=10.0.20.45

# The internal IP address of the syslog server
syslog_ip=10.0.20.67

# The admin password for the owncloud instance (login is "admin")
admin_pass=secret_password

# The version number of the ownCloud clode installed
OWNCLOUD_VERSION='6.0.2'

# Is this an enterprise version of owncloud? Set to true, it will install
# the enterprise version (the tgz file must be in 'roles/owncloud/files/owncloudEE)
enterprise=true

# is this a staging system? If true, the public keys in 'roles/common/files/dev_keys'
# are added to the 'authorized_keys' of the ubuntu user of the virtual machines
staging_system=true

# The email address that is being used to send LDAP statistics to
stats_send_to=owncloud-stats@example.com
stats_from=owncloud-stats@example.com

# server's mails to root (cronjobs) are sent to this address
notification_mail=drive-operations@example.com
```

Note - yes it's not DRY to list the various IP addresses again, as they could be computed from the hostsvars. However, for that to work, every single webserver playbook has to reach out to all the hosts that it would need the IP from. That makes the deployment of a single group of servers take longer. I have chosen to duplicate those IP adresses instead...

5.1.9 SSH Certificates

You will need a valid SSL certificate for the load balancer. The PEM file (with CRT and KEY concatenated) needs to be stored in `roles/haproxy/files/service_name.pem` where `service_name` is the name of the service as stored in the inventory file of Ansible (`production`, `staging`)

The LDAP server has a self signed certificate. To create that, run the following command:

```
ansible-playbook -i staging -s prepare_ssl.yml
```

This will create and distribute the LDAP certificate to the correct positions in the ansible directory structure.

5.1.10 Installation

Once everything is configured, run the site ansible playbook:

```
ansible-playbook -i staging -s site.yml
```

This will install the necessary software on all machines and configure them.

The Zabbix db server needs to be manually configured. Run these commands inside the Zabbix VM:

```
zcat /usr/share/zabbix-server-pgsql/{schema,images,data}.sql.gz | psql -h localhost zabbix zabbix
```

The actual Zabbix configuration needs to be done manually. In the directory `roles/zabbix/files/` there are three Zabbix exports that can be used as a starting point to configure the server.

You will also need to grant access to two tables in the owncloud database:

On the db server:

```
$ psql owncloud
grant select on oc_ldap_user_mapping to zabbix;
grant select  on oc_filecache to zabbix;

grant select on oc_ldap_user_mapping to nagios;
grant select  on oc_filecache to nagios;
```

CHAPTER

SIX

GARRBOX ARCHITECTURE

6.1 Service description

GARRbox is a personal cloud storage service designed for synchronizing and sharing users data in a secure and easy way. GARRbox is designed to support the needs of the Italian R&E Community. The service is the GARR response to a specific commitment from the Italian Ministry of Health for supporting the needs of the biomedical research community.

GARRbox takes inspiration from well-known cloud services like DropBox or Google Drive. It allows the synchronization of data among different devices like PC, smartphones and tablet, letting users share them with other users and collaboration groups in their home organizations.

GARRbox grants continuity and replica, adopting states of the art encryption techniques to protect data at rest.

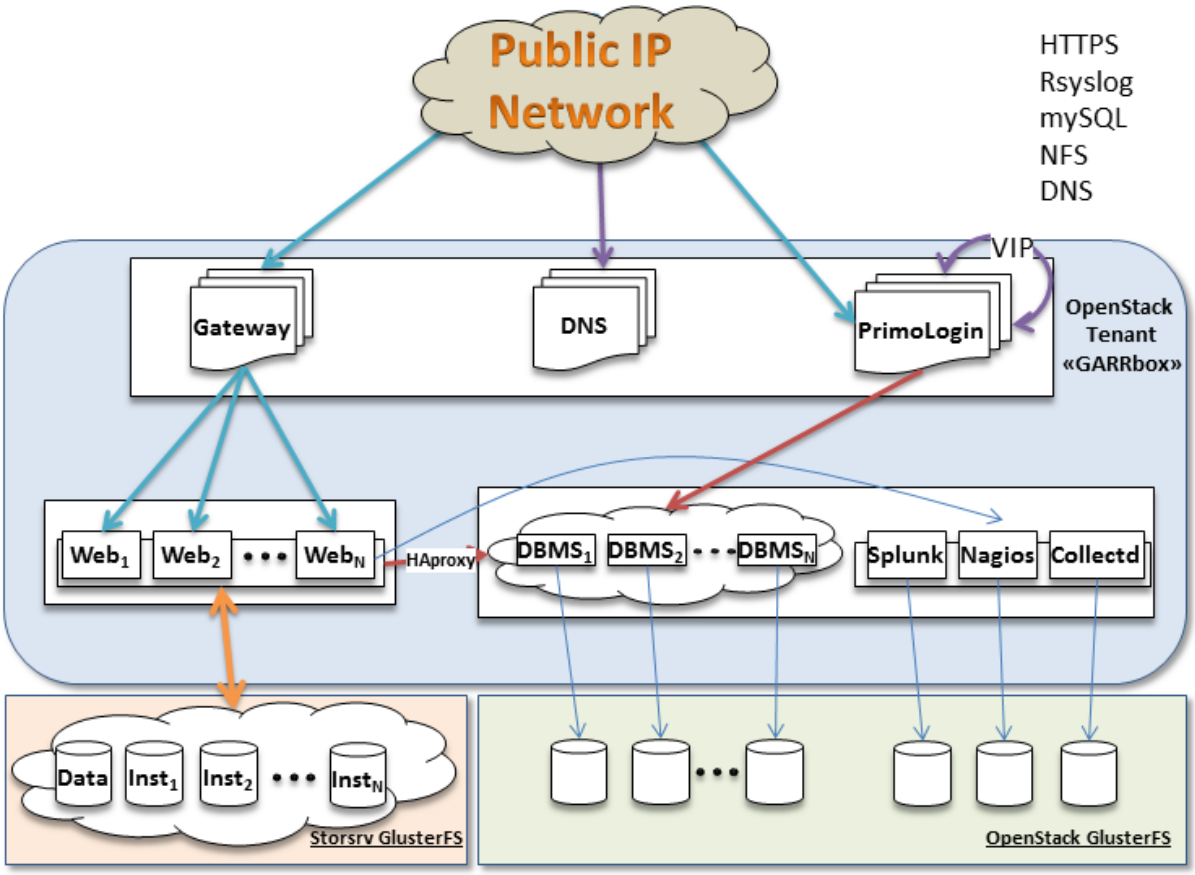
Users are not allowed to access the service directly. Their institutions subscribe the service and receive a number of users slots and a global amount of storage. By default every institution receives 100 user slots and 10 GB per user. The service supports autonomous administrative groups, so that the virtual managers at the institution can control user access and define individual quotas according the specific needs. When an institution needs more resources, they will be assigned with just a click.

GARRbox provides additional benefits to users

- It is designed to keep data inside the national borders and be run by national bodies, ensuring compliance with the laws on privacy, copyright and resilience.
- It is a joint effort of the Italian Research and Education community, supporting scale economies and equitable sharing of benefits among members.
- It relies on an ad hoc infrastructure, but thanks to the underlying cloud model principles it can scale in an elastic way to include more resources according to the current workload.
- It is operated by GARR. This is a guarantee for fairness, openness and compliancy to the standards and the best practices in both data preservation and sustainability of the service for many years to come.

6.2 Architecture

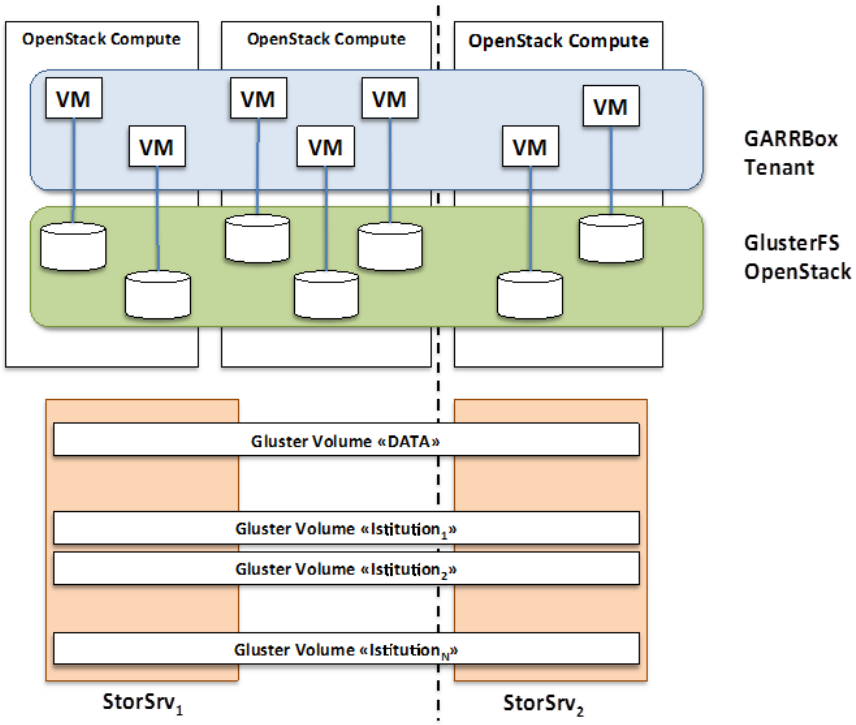
The following picture shows the current architecture diagram.



The current architecture has local HA (High Availability) enabled. All the VMs are running on physical hosts of the OpenStack deployment at GARR. The deployment is made of two sites at some kilometers from each other. The VMs storage (based on GlusterFS), the network (VLAN-based per tenant networks) and the control of the compute nodes are unified, so that the two racks operate as a single pod.

The users' data are hosted on a separate physical storage pool managed with GlusterFS.

The following picture shows a site view of the components layout.



All the VMs are under the aegis of a single tenant. VMs are grouped according their role in the architecture. For every role we defined a HA strategy and a virtual network. We also ensure that the VMs have been started on different compute nodes to minimize the impact in case of physical failure. In general, if one VM breaks, then there is at least another active VM that can provide the same are affected at the same time.

We defined the following roles.

- Access gateways and Name Servers
- Web/Application servers
- Database cluster
- Monitoring services
- PrimoLogin Authorization service
- Storage pools

6.3 Common

All the virtual machines are running Ubuntu in a standard server installation. Most of them run 12.04, while Web/Application machines have been recently upgraded to 14.04.

6.4 Access Gateways

Incoming requests are handled by Nginx acting as a caching proxy. Nginx does load balancing and SSL termination. Doing SSL termination on the load balancer allows it to inject cookies into the HTTP traffic that enables it to provide "sticky sessions" (i.e. requests from a single client always go to the same web server).

A second Nginx VM is configured in Active/Passive mode with virtual IP exchange in case of failure of the primary gateway through heartbeat.

The gateways log files and resource consumption information is sent to the monitoring services.

Access gateways use the following flavors: 4 vCPU, 4 GB RAM, 20 GB disk and are exposed to Internet.

Open Ports:

- 443

6.5 Name Servers

Name resolution is provided by two Bind instances configured in Active/Passive mode. The nodes are exposed to Internet and are built with the following flavors: 2 vCPU, 2 GB RAM, 20 GB disk.

6.6 Web/Application Servers

The web/application servers are handled by Apache and PHP5 FPM. Apache handles all incoming HTTP requests from the gateways using Event dispatcher, serves static files (css, js as well as the actual files stored in ownCloud) and passes all requests to ownCloud to the PHP5-FPM task.

PHP-FPM spawns a number of PHP worker processes on startup and manages them automatically (spawning more if the load rises and killing them again if load decreases). The goal is to have a PHP process ready whenever a request comes in. The process manager will always keep 8 PHP processes running and spawn up to 30 processes when under load.

Each web server mounts the ownCloud data directory from the storage pool (e.g., as `/opt/owncloud/data`). Inside that global mount point, storing the ownCloud entry points for the actual users' locations, there is an additional mount point for every volume assigned to a GARRbox subscriber. The data of the GARR constituents are kept separate. Users store data on their institution dedicate volume and are allowed to share data only among other members of the same institution.

The ownCloud logfiles and resources consumption are sent to the monitoring services.

The web servers are 4vCPU, 16GB RAM, 20GB disk virtual machines. The amount of RAM is so high because ownCloud allows large files to be uploaded through the web UI. The maximal file size is 2GB, but during the upload, the file is in memory. Lower amounts of RAM could lead to situations where the memory on the server is being exhausted.

For the first deployment we are using two active web server VMs and one back-up VM.

Open Ports (only to the internal network):

- 22
- 443

6.7 Database cluster

We decided to use Percona XtraDB Cluster instead of MySQL to store the database. The cluster is made of three nodes, two data nodes and the cluster coordinator, that ensure both HA and multi-master replication. The database is backed up periodically via a crontab entry.

As ownCloud is read heavy (factor 1000:1) the web/application VMs connect to the least loaded cluster node passing through local HAproxy load balancers running on the ownCloud nodes. Therefore, from the client perspective the whole cluster is a single endpoint.

The cluster stores also the Authorization service database. Also the Authorization service VMs access the cluster passing through the load balancer.

The VMs running the cluster are configured with 16 GB of RAM and 8 VCPUs and 20 GB storage.

Open Ports (only to the internal network):

- 22
- 3306

6.8 Monitoring services

All the VMs the Web/Application, Database cluster, Application gateway and Authorization subsystems send syslog data to the monitoring system.

In addition, the VMs are running probes to monitor the resources consumption.

The syslog entries are collected and indexed in a Splunk node for single events insights and, more in general, fine grain analytics. The information from the probes is collected at Nagios and Collectd virtual machines for system level checks and alarms on the status of the service.

We are evaluating open source alternatives to Splunk like LogStash.

6.9 Authorization Service: PrimoLogin

PrimoLogin (*first access* in Italian) is an auxiliary service developed by GARR both to bridge identities between AAI and local service credentials, needed by the synch clients on desktop and mobile, and to define and enforce authorization policies on institutions (groups of users), local virtual resources administrators and end users.

External users can login with AAI and are able to create a new account or to reset their password. According to the AAI affiliation, the access request is sent to the virtual resource administrators managing the pool of storage resources assigned to an institution. Through a joint usage of the features offered by ownCloud and Primologin, virtual resource administrators can grant access to their users, can block them and assign different personal quotas to users. Virtual resource administrators can also ask for larger collective storage pool or for additional slots when the allowed number of users is exceeded.

The server runs a Django application developed by GARR. GARR also developed a custom ownCloud login handler to enforce the policies that are defined by Primologin.

The VMs running PrimoLogin and its slave clone are equipped with 8vCPU, 8GB RAM, and 20GB disk. PrimoLogin is accessible from Internet.

Open ports:

- 443

6.10 Storage pools

GARRbox uses two storage sources, both exposed as GlusterFS volumes. Virtual machine disks are built on top of a replicated gluster volume mounted on all the OpenStack compute nodes to enable live-migration. Persistent storage used by Database Cluster and Monitoring VMs is provided by OpenStack Cinder using a distributed replicated striped volume.

For OwnCloud data we use a separate pool of storage resources. Every GARRbox subscriber has a dedicated volume with a quota that is negotiated at activation time. Usually, institutions get 1TB volume as initial resource. OwnCloud nodes mount the volumes as separate endpoints. Simple HA is implemented at mount point side using simplified virtual IP mechanism provided by CTDB.

User storage is backed up on a regular base.

6.11 Thoughts on High Availability at scale

The current setup aims at supporting a limited number of users with a minimal set of HA functionalities. The first phase of GARRbox aims at supporting biomedical research community, which typically has large sensible and valuable data like MRI images. Our current user community can involve up to about 5000 researchers.

Therefore, overall scalability is not the main goal in this moment. When we will become more confident with the service management, GARRbox will become available to larger communities like University researchers and, in a long term perspective to students.

We expect that the approach of having modular subsystems, most of which are stateless, with their own resiliency strategy allow us space for supporting quite easily larger user bases. Virtual machines failures are an infrequent event if compared to the underlying hardware failures. Live-migration, active/passive configurations and careful choice of VM initial locations (minimize the number of VMs on the same physical server) are a good starting point for getting both resiliency and balancing.

When larger user communities will access the service the architecture will have to satisfy both local scalability inside the datacenter, and global scalability among multiple sites at geographic scale. Storage clouds, because of the nature of the underlying virtual file-systems, are inherently constrained by the datacenters boundaries.

We believe that the only effective approach is the one adopted by big commercial players like Google or DropBox: have different datacenter with complete clones of the services, enact asynchronous replication of persistent information (data and databases) and access and failover control by using IP, DNS and load balancing.

The cloned VMs could be left in suspended status to avoid the loss of resources, and activated in

The adoption of proxies moving the frequently retrieved contents closer to the users is an additional optimization that will be taken into account.

CHAPTER

SEVEN

GARR INSTALLATION

7.1 Prerequisites

7.1.1 Software

GARR is currently evaluating Fabric as a supporting tool for system administration.

Fabric is a Python (2.5-2.7) library and command-line tool for streamlining the use of SSH for application deployment or systems administration tasks. It provides a basic suite of operations for executing local or remote shell commands (normally or via sudo) and uploading/downloading files, as well as auxiliary functionality such as prompting the running user for input, or aborting execution. Fabric is logically divided in two phases: configuration and deploy where in configuration phase we prepare the garrbox installation and we run a deploy phase for every production machine.

CHAPTER

EIGHT

CESNET SERVICE DESCRIPTION

CESNET's ownCloud is a relatively new service. It has been added to the CESNET Storage department's service portfolio on 11th of April 2014. The service operated by Czech NREN is meant to be an alternative to some well-known global cloud storage services. Its purpose is to support research & academic user groups with storage and collaboration needs.

Access to the service is allowed for institutions involved in a Czech national academic federation eduID (operated by CESNET). It's also possible to gain access to the ownCloud if the user doesn't have access to the federation, provided the user is affiliated to an institution compliant with the *CESNET Access Policy_*. For these situations, a validated Hostel account can be used to access the federated web interface (validated accounts require the users to prove their affiliation to an academic and/or research organizations by means of supplying a signed statement).

As of January 2015, there are 2600 users using the service with approx. 17 TB of stored data (14 millions of files). The most recent statistics of our ownCloud service can be viewed on our website.

Accounts for newcomers are created automatically upon first login and each new account is given a 100 GB quota. We are providing the service with a best-effort policy (i.e., we do our best to keep the service up and running, there is no formal SLA) and with no pay-per-use model. The users are required to comply with data storage general Terms of service.

Our ownCloud service can be used as:

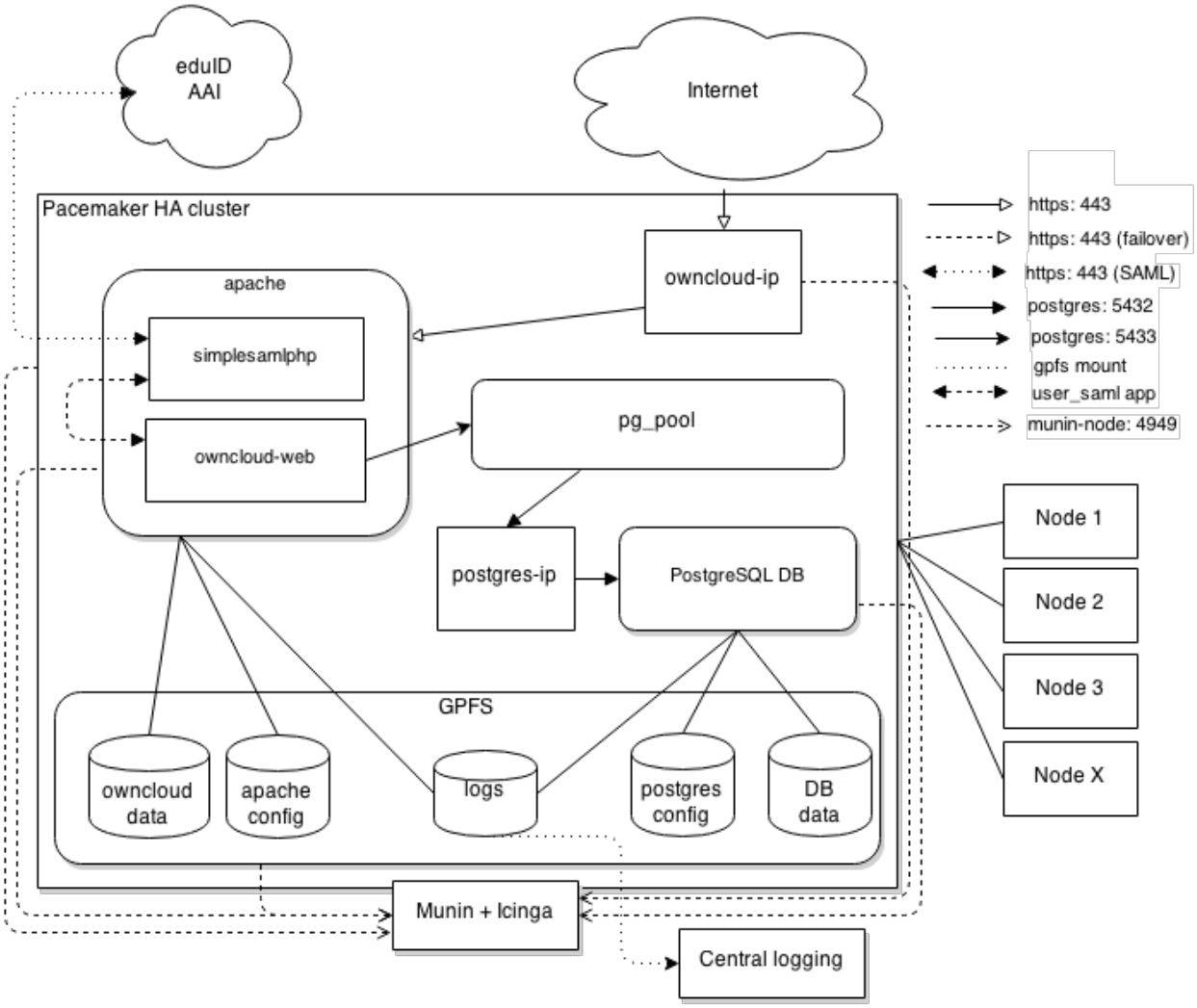
- collaboration platform (file sharing and collaborative document editing),
- local cloud storage platform (synchronization between clients, all data stored in the territory of the Czech Republic),
- backup platform (all stored data is backed up to tapes; more suitable backup solutions are nevertheless provided by CESNET Data Storage for backups of larger data sets).

CHAPTER

NINE

CESNET ARCHITECTURE

The following picture shows the current architecture of the ownCloud service at CESNET.



Our ownCloud architecture is built on top of HA (High Availability) cluster managed by Pacemaker. The cluster consists of 5 equivalent nodes. Pacemaker ensures that ownCloud and other relevant services are running properly even if one or more front-end dies. All nodes are physical machines, currently there is no virtualization. Key feature of our architecture is that all nodes are equal as each of them can take any role (be it an app server or a DB server).

Our system consists of the following components and services:

- Application server
- Database server
- PgPool proxy
- Pacemaker HA services
- Icinga & Munin monitoring servers
- eduID federated authentication service
- GPFS storage
- TSM server for backups

9.1 Specs

Hardware specifications of each front-end are as follows:

- CPU: 2 x Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz (24 CPUs)
- RAM: 96 GB (~ 40 GB ownCloud-dedicated)
- Network: 2 x 10 GE (bonded)

GPFS is being used as an underlying filesystem. OwnCloud utilizes a dedicated 40 TB volume which is mounted on all front-ends. The IBM DCS3700 disk array is being used for data storage.

All front-end nodes are running Red Hat Enterprise Linux 6.

9.2 Application Server

OwnCloud application is handled by Apache 2.2 and PHP 5.4 (which has been installed from non-standard repository – see this guide).

Newer PHP version was used instead of PHP 5.3 version (which is default on RHEL 6) to allow uploads of files larger than 2 GB. There were problems with this in 5.3.X series. We are able to do up to 16 GB uploads on PHP5.4 with no problems at all.

Apache is running as a standalone server without any HTTP(S) proxies in the way. It is listening on ports 80 and 443. HTTPS is being forced from 80 to 443 by redirection.

Complete Apache + PHP configuration and website root is located on a shared GPFS volume, which is mounted on all cluster nodes.

Apache is configured to use the default **prefork** MPM module for handling requests. However, its configuration has been tweaked to match nodes specs and current load:

```
<IfModule prefork.c>
StartServers         500
MinSpareServers       10
MaxSpareServers       30
ServerLimit         1800
MaxClients          1600
MaxRequestsPerChild 8000
</IfModule>
```

Since we have quite a lot of RAM available on the servers, we increased the maximum number of workers to 1600, so it uses up to 40 GB of RAM with some reserve (by our measurements, one Apache worker process takes 22 MB in average). The **StartServers** directive is set to a quite high value, too, in order to handle peak loads after server restarts. In such a situation, user's clients start reconnecting massively, overloading the server significantly for a short period of time.

Apache is set up with Zend OPcache module for better PHP performance and mod_xsendfile, which is being used for faster file downloads (also provides users with pause/resume download capabilities).

Open Ports:

- 80
- 443

9.3 Database Server

PostgreSQL 8.4 is being used as a database server of choice. This is the highest version available in standard RHEL 6 repositories. The database server runs in a single instance, preferably on a different node than the Apache instance and without any replication yet. We are planning an upgrade to 9.4 and a hot-standby streaming replication setup with load balancing. We expect this to help us in the future with a read heavy nature of ownCloud queries, but as of now the current setup copes with the load just fine.

OwnCloud PHP application doesn't access the PostgreSQL instance directly. It sends its queries through PgPool II, which acts as a connection cache (running in *Connection Pooling* mode). This gives us some performance boost as it reduces database instance load by means of reusing existing connections (thus saving cost of creating new ones). PgPool is run together with the Apache instance on the same node.

PostgreSQL engine parameters are mostly set according to the pgtune utility recommendations. It has been allocated 40 GB of RAM, the same as the Apache instance, and a maximum number of connections is set to **51** (mostly based on recommendations in PostgreSQL documentation).

The database server keeps its data and configuration on a shared GPFS volume just like the Apache server.

Open Ports:

- 5432
- 5433 (PgPool II)

9.4 High Availability

Pacemaker 1.1 on top of RedHat's CMAN 3.0 is used as High Availability resource manager for all services and applications. Main advantages of this setup are

- defined order in which individual resources are started,
- default locations where services are run,
- automatic distribution of services based on their mutual linkage,
- periodical checks if all services are running,
- and automatic failover.

We use basic Resource Agents (RAs) available from system repository for controlling Apache, IP aliases and PostgreSQL services. Due to the lack of repository RA for PgPool II, we developed our own RA script. From the ownCloud point of view, we use six resources, all of them run in active-passive mode. First of all, PostgreSQL DB engine is started on a node and an IP alias for database is configured. Start of the PgPool II RA is the next step, which takes

place on a different node than the DB. Both IPv4 and IPv6 aliases for ownCloud service are started on the node as soon as the PgPool II is running, and finally the Apache RA is started. Pacemaker guarantees shutdown of all services in a contrary order if necessary.

9.5 User Authentication

Authentication of users is based on SAML. It relies on the SimpleSAMLphp backend application for authentication and providing user's metadata. SimpleSAMLphp backend is configured with eduID IdP (Identity Providers) metadata and acts like an SP (Service Provider) in the federations. When users try to log in, they are presented with a WAYF page, where they can pick their home organizations. They are then redirected to their organization's IdP login page where they log in. After a successful log in, we get all necessary information about a user (EPPN, e-mail) from user's home organization IdP.

When we were looking for a solution of user authentication, there were two available user backends for ownCloud, which allowed federated user accounts to log in – user_saml and user_shibboleth. Both of them were quite outdated and not working well in ownCloud 6, however. We have picked the *user_saml* app and fixed an issues it had with OC 6 in this pull request. Without those fixes, user creation and logout was broken. That way only already existing ownCloud users could log in using SAML authentication and the 'Logout' option from the menu did nothing.

9.6 Data Storage and Backup

All the data is stored in a dedicated GPFS filesystem mounted on all nodes, so all nodes in the cluster can access the same data. For this filesystem, we reserved 40TB of disk space. The filesystem is built on top of 4 RAID6 groups from IBM DCS3700 disk array, which is connected through Fibre Channel infrastructure to all frontend nodes. We use this filesystem to store PostgreSQL database datafiles, ownCloud user data, web interface files for the webserver, as well as logging of all installation components.

Data backups are realized by a GPFS utility mmbackup. This utility scans the whole filesystem (using GPFS inode scan interface) and passes a changed, new or deleted files to TSM (Tivoli Storage Manager) server. TSM then runs selective (full) backup (or expiration when file deleted) on those files. We retain a history of 2 versions of the backed files in TSM for 30 days. TSM is being used with IBM TS3500 tape library as a persistent storage device for holding backups. OwnCloud backups are run periodically once a day.

Before each backup run, PostgreSQL database is being dumped using pg_dump utility. Pg_dump generates the archive and mmbackup then finds this file on the GPFS filesystem and sends it to TSM with the rest of ownCloud files to be backed up.

9.7 Monitoring

All ownCloud specific services are constantly monitored by Icinga (a fork of Nagios). We had to write own custom plugins to check some ownCloud specific stuff. Following items are being periodically checked by Icinga:

- SSL certificate validity
- WebDAV client functioning properly (data can be uploaded and downloaded)
- Free space on OC GPFS volume
- Apache responding on HTTPS
- PING (machine with owncloud-ip responding)
- PostgreSQL (Postgres is responding on postgres-ip and OC can connect to the database)

In addition to this, we use custom Munin plugins to collect usage statistics and create graphs. Currently we are graphing the following ownCloud statistics:

- Number of user accounts
- Number of files
- Amount of user data stored
- Apache response times
- Bytes transferred by Apache
- Filesystem space used

We are also collecting all relevant logs to a central server, where it could be further analyzed and queried with LogStash and ElasticSearch.

CHAPTER

TEN

CESNET INSTALLATION

10.1 Prerequisites

In order to replicate our ownCloud deployment, you will need the following software prerequisites to be installed on your designated nodes:

- Puppet 2.7.x
- Pacemaker
- Munin-node
- GPFS
- Nrpe + nagios-plugins-nrpe
- RHEL6 or Debian

For monitoring and reporting purposes, you should also have some dedicated servers at your disposal with Nagios (Icinga) and Munin servers installed.

10.2 Preparing storage

You should start by deciding where to place the whole ownCloud installation.

We have decided to place everything ownCloud related on a shared filesystem mounted on all nodes. This allows achieving High Availability, Pacemaker is then able to move services between the nodes freely if one of the nodes fails. GPFS shared filesystem creation and mounting is described on the IBM wiki. We created a filesystem named *owncloud* using the following stanza file:

```
# owncloudfs.stanza
# This assumes you have already created NSDs
# (mmaddisk) from your disk array LUNs
%nsd:
        nsd=name_of_nsd
        usage=dataAndMetadata
...
```

And then executing the following commands (this will assign NSDs to the newly created filesystem):

```
mmcrfs owncloud -F owncloudfs.stanza -A no -B 2M -D nfs4 -n 5
mmmount owncloud -a
```

You should now have the filesystem prepared and mounted on all nodes in the GPFS cluster.

10.3 Puppet

Puppet client has to be installed on all nodes participating in the ownCloud cluster. All packages needed could be obtained from the Puppetlabs repository. You will need to install some Puppet version < 3.0, because the modules we use haven't been tested with *Puppet 3.x* versions yet.

There are two modes available when deploying the Puppet – **standalone** client or **master/agent**. Our Puppet configuration is tested with the master/agent setup, but it should work fine even when using just a standalone clients. You can read more about these two modes and how to deploy Puppet in the Puppet installation guide.

With Puppet installed and properly configured, download the following puppet modules:

Apache module:

```
https://github.com/CESNET/puppet-apache #(to be uploaded)
```

PostgreSQL module (including PgPool II):

```
https://github.com/CESNET/puppet-postgres #(to be uploaded)
```

Put these two modules into the 'modulepath' (`/etc/puppet/modules/` by default) of Puppet. Both modules are then linked together and initialized by an ownCloud profile inside the 'nodes.pp' manifest. We then assign the profile to the ownCloud cluster nodes.

Note: We use Puppet's Hiera for storing the variables, but you can avoid using Hiera by substituting all `hiera()` calls with variable values.

The relevant part of our 'nodes.pp' manifest follows:

```
#/etc/puppet/manifests/nodes.pp
class profile::owncloud (
  $base_dir  = '',
  $log_dir   = hiera('owncloud::log_dir')
) {

  # Configure PostgreSQL services
  # -----------------------------
  include ::postgres::backup

  class { '::postgres::server':
    conf_dir   => hiera('postgres::conf_dir', "${base_dir}/pgdata/"),
    listen_ip  => hiera('postgres::listen_ip')
  }

  class { '::postgres::pgpool':
    conf_dir         => hiera('postgres::pgpool::conf_dir',
                              "${base_dir}/etc/pgpool"),
    backend_hostname => hiera('postgres::listen_ip'),
  }

  # Configure Apache services
  # -------------------------
  $modpkgs = ['mod_ssl','mod_xsendfile']
  $config  = 'apache2/etc/httpd/httpd_oc.conf.erb'

  class { '::apache2::server':
    base_dir        => $base_dir,
    httpd_source    => $config,
    enabled_modules => ['ssl', 'xsendfile', 'rewrite'],
```

```
    disabled_sites  => ['default', 'default-ssl'],
    module_pkgs     => $modpkgs,
    manage_service  => true,
    reload_cmd      => $reloadcmd,
    oldlogs_dir     => "${log_dir}/old-logs/"
  }

  class {'::apache2::php':
    extension_packages  => [
      'php54', 'php54-php',
      'php54-php-cli', 'php54-php-common', 'php54-php-devel',
      'php54-php-gd', 'php54-php-mbstring', 'php54-php-pdo',
      'php54-php-pear', 'php54-php-pgsql',
      'php54-php-process', 'php54-php-xml', 'php54-runtime',
    ],
    php_module          => 'modules/libphp54-php5.so',
    post_max_size       => '16G',
    upload_tmp_dir      => "${base_dir}/tmp",
    upload_max_filesize => '16G',
  }

  include ::apache2::simplesamlphp

  class { '::apache2::owncloud': webdir => hiera('owncloud::webdir') }
}

node /your-node.hostnames.com/ {
  class { 'profile::owncloud': base_dir => '/yours/gpfs/mountpoint' }
}
```

When using Puppet in a standalone mode, issue the following command on each node:

```
# puppet apply /etc/puppet/manifests/nodes.pp
```

If you are running in the master/agent mode, deployment will be done automatically by Puppet agents. This way you should now have all the ownCloud specific services deployed to all nodes.

10.4 Pacemaker

The basic installation of Pacemaker HA manager on RHEL 6 system is not covered in this text and can be found elsewhere. In this section, let us assume that fully functional Pacemaker is installed on at least three hosts with working STONITHd and all necessary dependencies like, e.g., filesystem resources. We also assume that all necessary RAs are have been installed as part of the Pacemaker installation and placed in /usr/lib/ocf/resource.d/. The only remaining RA is the one for controlling PgPool II. This RA needs to be written, or, more conveniently, *CESNET version_* can be downloaded.

All examples of Pacemaker configuration are meant to be used with the help of crmshell. Service definition may be the following:

```
primitive PSQL_OC pgsql \
op monitor interval=60s timeout=30s on-fail=restart \
op start interval=0 timeout=600s on-fail=restart requires=fencing \
op stop interval=0 timeout=120s on-fail=fence \
params pgdata="/some_path/pgsql/data/" pghost=IP_address monitor_password=password monitor_user=user
meta resource-stickiness=100 migration-threshold=10 target-role=Started
```

Special database monitor is used for the monitoring of the PostgreSQL database. We recommend keeping minimally one connection to the database unhanded by PgPool II so this monitor can use it. Another example is definition of PgPool II service based on our RA:

```
primitive pgpool-owncloud-postgres ocf:du:pgpool_ra.rhel \
params pgpool_conf="/pgpool_inst_path/etc/pgpool/pgpool.conf" pgpool_pcp="/pgpool_inst_path/etc/pgpoo
meta resource-stickiness=10 migration-threshold=10 target-role=Started \
op monitor interval=60s timeout=40s on-fail=restart \
op start interval=0 timeout=60s on-fail=restart requires=fencing \
op stop interval=0 timeout=60s on-fail=fence
```

All remaining necessary services are configured in the same manner. Suitable parameters of different RAs can be tested by direct running of those scripts. For example the database can be monitored by this command:

```
OCF_ROOT=/usr/lib/ocf OCF_RESKEY_pgdata="/some_path/pgsql/data/" OCF_RESKEY_pghost=IP_address OCF_RES
```

Next all location, colocation, and order linkages must by specified. In order to achieve our configuration as described in architecture file next lines must be added into Pacemaker configuration:

```
location l-PSQL_OC-fe4 PSQL_OC 600: fe4-priv
location l-PSQL_OC-fe5 PSQL_OC 500: fe5-priv
location l-owncloud-web-fe4 owncloud-web 500: fe4-priv
location l-owncloud-web-fe5 owncloud-web 600: fe5-priv
```

Location statement determines on which nodes the service should start with some priorities. Colocation is used to specify which services should be started together on the same host and which must be on different hosts. Each colocation has appropriate weight or inf and -inf are used for absolute meanings. Resolving of the colocation dependencies is performed from right to left.:

```
colocation c-FS-services inf: ( PSQL_OC owncloud-web ) FS
colocation c-PSQL_OC-IP inf: PSQL_OC PSQL-ip
colocation c-owncloud_web-IP inf: owncloud-web owncloud-ip owncloud-ipv6 pgpool-owncloud-postgres
```

So the last rule means that owncloud-web is run where owncloud-ip is running and that is on the same node as owncloud-ipv6 and that is where pgpool-owncloud-postgres service is running. The last parameter changes the order in which are services started and stopped.:

```
order o-FS-services inf: FS ( PSQL_OC owncloud-web )
order o-PSQL-Owncloud_web inf: PSQL_OC pgpool-owncloud-postgres owncloud-web
order o-PSQL_OC-IP inf: PSQL-ip PSQL_OC
order o-owncloud_web-IP inf: owncloud-ip owncloud-web
order o-owncloud_web-IPv6 inf: owncloud-ipv6 owncloud-web
```

After successful configuration of all services, fine tuning of timeout values is necessary according to overall behaviour of the system. There are no general values of timeouts, we recommend to start with the ones from RA scripts.

10.5 Setting up ownCloud

In the next step, you will need to download and install ownCloud from the source archive. Just follow the Download ownCloud and Set permissions sections of the official installation guide. Just put it in a directory specified in the Puppet's 'nodes.pp' variable 'webdir'.

For the user SAML authentication to work properly, you need to fetch the 'user_saml' app from the owncloud/apps GitHub repository. It already contains our fixes of the 'user_saml' app. If you are interested in our modifications as described in the *CESNET Modifications to ownCloud* chapter, you can use the cesnet/owncloud-apps repository instead.

Then you create 'owncloud' DB table and user and go through the standard ownCloud webinstall. When you are done with installation, it is important to note the **instanceid** generated by ownCloud. You can find it in the ownCloud's 'config.php'. It will be needed by the SimpleSAMLphp.

10.5.1 SimpleSAMLphp

Now you'll need to finish the configuration of an authentication backend used by the 'user_saml' app. Most of the things should be already put in place by Puppet, but you will need to have a look and modify some files referenced by the 'apache2::simplesamlphp' Puppet class. In 'config.php.erb', you will need to change the cookiename to the 'instanceid' noted in the section before:

```
'session.phpsession.cookiename'  => 'oc1234567',
```

In the 'authsources.php.erb', change the attributes in the 'default-sp' section according to your environment:

```
'default-sp' => array(
        'saml:SP',
        'entityID' => 'https://<%= @oc_hostname %>/saml/sp',
        'idp' => NULL,
        'privatekey' => '<%= @key_dir %>/<%= @oc_hostname %>.key',
        'certificate' => '<%= @cert_dir %>/<%= @oc_hostname %>.crt',
        // eduID.cz + hostel WAYFlet
        'discoURL' => 'https://ds.eduid.cz/wayf.php...'
```

The last thing needed is to specify sources of IdPs (Identity Providers) metadata. This can be done in the 'config-metarefresh.php.erb' file:

```
'eduidcz' => array(
        'cron'          => array('daily'),
        'sources'       => array(
                array(
                        'src' => 'https://metadata.eduid.cz/...',
                ),
        ),
        'expireAfter' => 60*60*24*4, // Maximum 4 days cache time.
        'outputDir'   => 'metadata/eduidcz/',
)
```

Metadata are then refreshed periodically by a cron job already installed by Puppet. More generic information about setting your own SP in the SimpleSAMLphp could be found in the official configuration guide.

10.5.2 User_saml Configuration

The last step needed to get the user authentication running is to enable the 'user_saml' app in the ownCloud and configure it properly. This can be done in the web administration interface. You just need to set a proper paths to SimpleSAMLphp and user's attribute mapping from SimpleSAMLphp to ownCloud according to your environment. You can test which attribute names SimpleSAML gives you about a user on its testing page:

```
https://your.domain.com/simplesamlphp/module.php/core/authenticate.php?as=default-sp
```

CHAPTER

ELEVEN

CESNET MODIFICATIONS TO OWNCLOUD

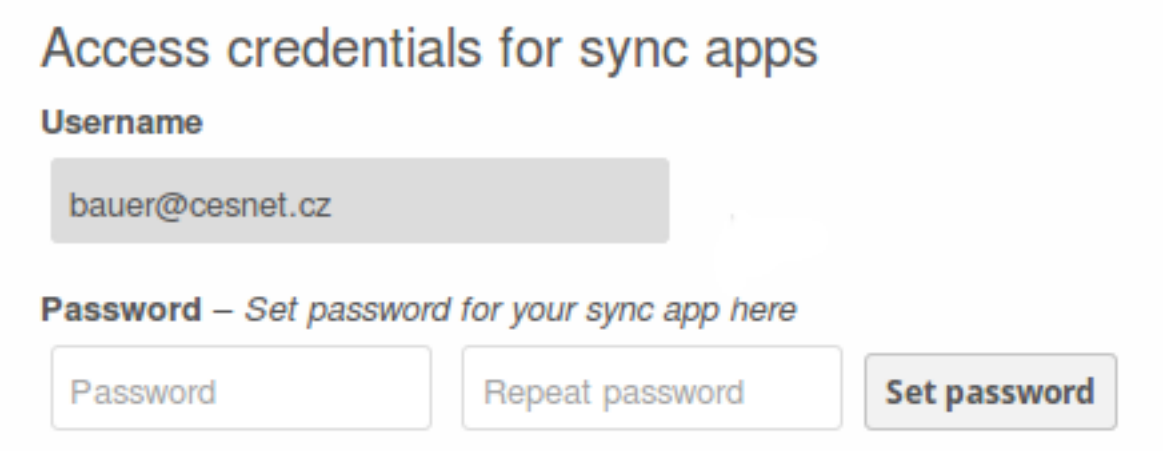
We aren't running just a stock ownCloud source with some 3rd party apps installed. During the time we provided our sevice, we came across some specific requirements, so we needed to develop custom modifications. They are available at our cesnet/owncloud-apps fork. This chapter describes some of the modifications needed.

11.1 User WebDAV Authentication

In our setup, the 'user_saml' app is responsible for the task of authenticating and managing the user accounts. Fixing it as discussed in the *User Authentication* section enabled users to log in and automatically create accounts.

But there were still problems that needed to be solved. Automatical account creation had a side effect. Since users are entering their login credentials at their organizations IdP, ownCloud has no way knowing the user's password. That way, the 'user_saml' app must set a long enough random password for each new ownCloud user automatically created.

Users don't need to know their ownCloud password as long as they don't use some WebDAV client. But clients need the ownCloud password and this renders them completely unusable. To solve this, we have modded the 'user_saml' application so it adds extra fields to each user's 'Personal Settings' page, so that they can configure their clients properly.



11.2 Account Consolidation

Since we are operating the service with federated accounts, it could easily happen that some users have multiple accounts at multiple organization IdPs. Normally, we would create an independent ownCloud account for each organization's account the user has. But with consolidation, it is possible to map all user's IdP accounts to a single

ownCloud account. So when the user leaves one organization and his account there is closed, he can still access the same ownCloud data using the second IdP's account he may have.

Note: This topic is still a 'work in progress' for us, as we have just a basic support for this feature implemented. We are going to collaborate with the Perun AAI developers and use Perun's consolidation mechanisms.

11.3 File Sharing

There were several improvement requests to the ownCloud 6 sharing mechanisms that we implemented ourselves (they were just too specific to our environment to be considered suitable for a pull request to upstream).

By default, the share's autocomplete function offers users with usernames (and full names) of the other users as they start typing. In our environment, this has proven to be undesirable as our users cannot choose usernames. It is determined by the EPPN identifier sent to us by the user's IdP. In some organizations, an EPPN is considered to be a sensitive information, so we were asked to disable the autocomplete function.

On the other hand, disabling autocomplete function would worsen the user experience when adding shares. So we implemented an autocompletion from the user's contacts at least. You just need to import contacts you want to quickly share files to in the 'Contacts' app.

We plan to integrate group management in ownCloud with our user management system Perun. Perun defines groups and handles user membership in them, it would be convenient for the users to be able to use the groups in all systems we operate.

CHAPTER

TWELVE

CESNET EXPERIENCES

This chapter discusses technical challenges we have faced and experiences we have gained while deploying and running the ownCloud service.

12.1 Deployment

When we started deploying the ownCloud service, Puppet made our lives much easier. We were benefiting from an already existing Puppet infrastructure that is used to manage the whole storage infrastructure and some basic Puppet modules from which we could build the service. It would certainly be much more uncomfortable experience to configure and deploy all pieces needed to each node manually (or in an 'ssh in a for loop' way).

Puppet automated most of the deployment tasks for us. We developed modules for each service, then specified to which nodes to deploy the services and watched Puppet do its job. With this approach, we are able to easily add more nodes to our ownCloud cluster or to deploy the same setup onto a different site.

12.2 HA Configuration

The only major things we kept away from the Puppet's reach were filesystem preparation and a Pacemaker HA configuration. Preparing a filesystem was pretty straightforward, but setting up a HA cluster has proven to be a major challenge for us. We had to prepare RA's (Resource Agents) for some component desired to run in HA mode. As the number of services managed by Pacemaker grown up in time, the whole configuration of proper linkage between services and hosts become more and more complex. The most time consuming and problematic part was the fine tuning of timeout parameters of basic actions of all services. In normal non stressful conditions setted up timeouts were too low in some stress conditions. This behaviour led to strange and difficult to identify restarts of services and in some cases whole nodes. But nowadays, after hours and hours of tuning, we can say, that it's pretty stable and behaves the way we expect it to.

12.3 ownCloud Application

For the time we were running our ownCloud service, we have stumbled upon a handful of interesting software problems / bugs, both in ownCloud and synchronization clients. Some of them are described in the following section.

12.3.1 User Files Coming from the Future

Among the first things we observed (and we are still observing) in our logs were errors like this:

```
{...,An exception occurred while executing
UPDATE "oc_filecache" SET "mtime" = ?, "etag" = ?, \"storage_mtime\"=?
WHERE "fileid" = ?: SQLSTATE[22003]: Numeric value out of range: 7
ERROR:  value \"2497698996\" is out of range for type integer"...}
```

Root cause of these errors seemed to be that files with bad mtimes were synchronized to us from the user clients. Many of those files were coming from a cameras without a properly set date, or any other misconfigured devices. This caused sync problems and even unability to log in for the one of the users. But right now we don't have a better solution other than to recommend users to watch out for these files. The issue is currently being discussed in this ticket as a bug.

12.3.2 Cron Job Traffic Jam

For a long time, we have been fighting with a steadily increasing ownCloud website response times. It got to the point where it was a way too sluggish at a given number of users and usage. When looking at the page loading times, it took most of the time to load generated JS files like 'core.js'. But after some XDEBUG profiling we started to suspect the database to be the major bottleneck. Database node had all CPU cores utilized to the max almost all the time. The response times of database backend were way too high. PgPool II deployment and PostgreSQL configuration tuning helped for a while, but after few months it got worse again.

One could think that this could be attributed to a big gain of users in October 2014 as you can see in our ownCloud statistics. But when we started looking at a queries that were being executed on the database, we saw that almost **90%** of them were just SELECTs on the 'oc_jobs' table:

```
SELECT `id`, `class`, `last_run`, `argument` FROM `oc_jobs` WHERE `id` > ? ORDER BY `id` ASC
```

The 'oc_jobs' table also counted more than 80,000 rows. This was a clear indication that the ownCloud's cron job (cron.php script which is supposed to delete old temporary files and clean the 'oc_jobs' and other DB tables) didn't do its job properly. Cron job was configured to be executed by the system cron. It was executing every 15 minutes, but it did nothing.

Then we realized that this was because of an existing lockfile placed inside ownCloud's data folder. This must have been left there for a long time by some previous cron job run (which must have ended really badly). We however didn't isolate the source of those SELECTs. But after we removed that lockfile and ran the cron job again, there were only 2 records left in the 'oc_jobs' table. Since then, ownCloud's overall performance went dramatically up and is up to our expectations. We are now serving more than 2600 users still on a single Apache & DB instance without any load balancing. And we are expecting to serve even more using this setup for the following months.

CHAPTER

THIRTEEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*