Computational
Health
Informatics

Leibniz
Universität
Hannover

Lab: Linux System Administration

Summer semester 2025

Dr. Hans Georg Krojanski

2025-07-07

Contents

1 Introduction

→ Introductory lecture

2 Linux and shell basics

→ Slides in Stud.IP

3 Basic server security

3.1 nftables

`nftables` is a Linux kernel framework to classify and "mangle" packets. It can be used to implement e.g. a packet filter firewall, network address translation (NAT) or packet forwarding. The command line utility to manage this is `nft(8)`.

`nftables` replaces the legacy "xtables" infrastructure (→ iptables, ip6tables, arptables, ebtables).

This talk (video and slides available in german) is a good introduction to nftables: Einführung in nftables

You will use `nftables` in this lab to setup packet filtering on every VM, using the following principles:

- Allow every outgoing traffic (we already discussed if certain ports/protocols should be disallowed to improve security, but we'll start with this setup)
- Do not allow any incoming traffic other than:
  - Already established traffic
  - Traffic to ports where a (necessary) service is listening

But even in this last case, always think about from where the traffic should be allowed; e.g. only from internal systems or from outside (the internet).

Very important advice: It is very easy to accidentally deny yourself access to a server when configuring a firewall. In this case, you may not be able to revert the last changes, at least not without precautions. You can reduce this risk if you write a shell script to apply new rules which has some safeguards. A step-by-step example of writing such a script with detailed explanations can be found here: Safe reload with nftables

- Install nftables and configure it (→ nftables Wiki) on all VMs
- Do not use legacy `xtables`, do not use `firewalld` or any similar software on top of `nftables`
- Use `systemctl status` to get information about and the status of the nftables service
- What is the path of the service file for `nftables` that is currently loaded?
- Look at the contents of this service file. What are the commands used for …
  - … `systemctl stop nftables.service`?
  - … `systemctl start nftables.service`?
  - … `systemctl reload nftables.service`?
  - … `systemctl restart nftables.service`?

- Remember not only to start the corresponding `systemd` service for `nftables`, but also to enable it for automatic start after a reboot
- Find out the IP address of the jumphost VM (which is **not** the IP of the ”studserv” server itself). How did you achieve this?
- Allow ssh login to **ll-admin** only from this jumphost IP
- Allow ssh login to the other VMs only from **ll-admin**

Don't forget to allow other ports if you're installing other services like a web server or database. But always take into consideration from which IPs or IP ranges access to theses services is really needed. It is also advisable to use different firewall rules for operations and for testing purposes in order to enable or disable them independently.

3.2 sshd

`sshd(8)` is already running on all VMs. In the current configuration, the login is possible with passwords.

- Important precaution when making changes to the ssh service on a remote (i.e. not otherwise accessible) system:
  - Split your `tmux` window to get another pane
  - In this new pane, run temporarily another ssh process on port 2222, but take care that this does not run as a daemon (which `sshd(8)` option does that?)
  - Verify that you can login via this second ssh port 2222; you'll need this (only) in an emergency if the usual `sshd` is not working anymore
  - Don't forget to open the firewall (temporarily) for this new ssh port
- Use `ssh-copy-id(1)` to copy your ssh public key, that you already created for this lab, to **ll-admin**. Which command options do you need for this (e.g. to use the `ProxyJump` option passed through to `ssh`, which `ssh-copy-id` uses)?
  Tip: Use an *additional* option so that `ssh-copy-id` doesn't check if the keys are present on the remote server.
- Test if the login with your ssh key works
- If it does, configure `sshd` to *only* allow login with ssh keys
- Use the `sshd(8)` test mode to check the validiy of the config files before restarting the daemon. What is the output if there is an error in the config file? What is the output if there isn't (how can you verify this)?
- Output the *effective* ssh configuration to stdout, including default values; redirect this to a file and answer the following questions:
  - Which port is used?
  - Is root login allowed?
  - Are empty passwords allowed?
- Try to login with the password, not with the key (how can you *enforce* password usage on the client?). What error message do you get? Try running `ssh` with maximum verbosity.
- How can you see *after* the login *on the remote system* from which IP the connection was made and which ports (remote and local) are used?

- If everything works, don't forget to stop the temporary ssh process listening on port 2222 (how did you do that?) and close this port on the firewall again
- Finally test if your settings and configurations are still working after a `reboot` of the VM

For the rest of the lab tasks, it is convenient to have a ssh key pair on **ll-admin** and to use this to login to the other VMs.

- Create a new ssh keypair on **ll-admin** and copy the corresponding public key to all the other VMs
- Test if the login with this new ssh key works from your admin VM to all the other VMs
- Configure the `sshd(8)` daemons on all VMs to allow only logins with ssh keys
- Test that the login with ssh key still works on all VMs
- Test copying a file from **ll-admin** to another VM with `scp(1)`
- How could you copy a file from your laptop through the jumphost and through your admin VM directly to e.g. **ll-web**?

When using ssh, it is very convenient to write all the options necessary for a connection in a `ssh_config(5)` file.

- Write a `$HOME/.ssh/config` file on **ll-admin** to use simpler commands to log into the other VMs:
  - `ssh ll-web`
  - `ssh ll-app`
  - `ssh ll-db`
- Set the ssh port and ssh key file explicitly, even if it may not be necessary
- Specify that your ssh key is automatically added to the `ssh-agent(1)` as if done by running `ssh-add(1)`
- Is the `ssh-agent(1)` even running? If not, how can you do this and ensure that this is done automatically after a reboot?
- Configure your login to **ll-admin** from your laptop by using a `ssh_config(5)` file. Use two different host specifications in this file, one for the jumphost connection and another one for the VM login.

One possible source for errors when trying to login via ssh are the file permissions on the client side. What are the recommended or necessary file permissions for ...

- ... the `$HOME/.ssh/` directory?
- ... the `$HOME/.ssh/config` file?
- ... the `$HOME/.ssh/authorized_keys` file?
- ... the `$HOME/.ssh/PUBLICKEY` file?
- ... the `$HOME/.ssh/PRIVATEKEY` file?

Change one of the *necessary* file permissions and report what the corresponding error messages are if you are trying to use ssh in this case. (Don't forget to revert this afterwards.)

4 Webserver, TLS and CA

4.1 Webserver: HTTP

Before we begin with the container tasks and learn how to deploy web applications with containers, we will learn in this chapter how to setup our own certificate authority (CA) and use this to sign web server certificates to secure and verify the HTTPS connection. Later in this lab, we will also sign certificates which are used for other TLS connections, e.g. to a database or from the reverse proxy to the application itself.

Install the `nginx` software on **ll-web** and make sure the default page is accessible over HTTP via the following methods:

- From **ll-web** itself using `curl(1)`
- From **ll-admin** using `curl(1)`
- From your computer/laptop via the SOCKSv5 proxy using `curl(1)`
- Repeat the three tests above with ”netcat”
- From your computer/laptop via a SOCKSv5 proxy using a browser. Use the fully qualified domain name for this, not the IP of your server (resp. VM). (Refer to the introductory talk for information and further questions regarding the use of a SOCKSv5 proxy for testing.)

4.2 Webserver: HTTPS

Now that the webserver is reachable via HTTP, reconfigure it to activate HTTPS. You can find general information about that in the nginx documentation.

4.2.1 Snakeoil Certificates

1. Which package do you need to install on debian in order to get an auto-generated ”snakeoil” certificate (and corresponding private key)? **Hint:** Look at the various config files of nginx that are installed with the nginx package.
2. Install this ”snakeoil” certs package and find out where the cert and key are located. What are their file permissions, owner and group?
3. Activate HTTPS for nginx by using the already installed config files, including the snippets file to use the ”snakeoil” certificate
4. Use the same tests that you used for HTTP above to see if HTTPS works properly. Does your browser complain about something?
5. Use `curl(1)` to get the SSL/TLS certificate of the webserver and save it in a file.
6. Inspect this certificate file with `openssl(1)` and answer the following questions:
   - For which hostname is this cert valid?
   - What is the meaning of the certificate extension where the hostname appears again?
   - How long can the cert be used and who issued it?
   - From which config are all these values set during generation of the snakeoil cert? (For this, openssl is not needed.)

7. Configure nginx to automatically and permanently redirect from HTTP to HTTPS and test this behaviour afterwards
8. Configure nginx to use secure protocols and ciphers. One possible ressource that might help you is the mozilla ssl configuration generator.

4.2.2 Creating a private CA

Now, instead of using self-signed certificates, create your own private CA on **ll-admin** and sign a *certificate signing request (CSR)* for your webserver.

1. In general you can follow the guide from the Ubuntu server documentation but there are a few things that you should keep in mind when configuring the CA:
   - Set the *copy_extensions* option so that certain values, like *subjectAltName*, are not ignored when signing a CSR. Read the section about *copy_extensions* and the WARNINGS section in the `openssl-ca` manpage.
   - Set `unique_subject` so that several valid certificate entries may have the exact same subject (This is convenient for testing, e.g. when iterating a CSR until the corresponding signed cert is correct)
2. How do you prevent your CA from signing CSRs where the option `basicConstraints` is set to `CA:TRUE`?
3. Expore the **CA cert** with `openssl-x509(1)`. Are all entries as they should be? Copy the full certificate information (not the cert itself!) into your markdown documentation.

4.2.3 Generating a CSR

After setting up your CA you can now generate a CSR on each machine. For this you can also use the guide mentioned above, but please note the following tips:

- Specify the values for the CSR in a config file and use that, especially when you need to generate multiple CSRs.
- When specifying the address (no matter if in the CN, the *subjectAltName* or elsewhere) always append the TLD (e.g. '.incus'). This ensures that name resolution works correctly in all cases and that the address in the certificate always matches the address used to connect to a server.
- Set a *subjectAltName* extension. This is needed by some applications, like browsers, to actually accept the certificate. See the manpages `openssl-req` and `x509v3_config` for how to accomplish this. In addition, listing 1 shows a template for a certificate signing request config file.

Listing 1: CSR template file for ll-web-20

```
[ req ]
default_bits          = 4096
default_keyfile       = server-key.pem
default_md            = sha256
distinguished_name    = req_distinguished_name
req_extensions        = v3_req
# string_mask           = nombstr # DO NOT SET THIS or at least set this to the same
    value as in the CA config
```

```
prompt                  = no

# Some of the following settings have to match with your CA:

[ req_distinguished_name ]
countryName             = DE
stateOrProvinceName     = Niedersachsen
localityName            = Hannover
organizationName        = Leibniz Universitaet Hannover
organizationalUnitName = LinuxLab
commonName              = ll-web-20.incus
emailAddress            = krojanski@chi.uni-hannover.de

[ v3_req ]
basicConstraints        = CA:FALSE
keyUsage                = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName          = @alt_names

[ alt_names ]
DNS.1 = ll-web-20.incus
```

1. Which command did you use to generate a CSR on one of your machines?
2. Check with `openssl` that the generated CSR is correct and matches your config file entries. Which command did you use? What was the output?
3. Check the integrity (consistency) of the *private key* with `openssl`. Which command did you use? What was the output?
4. Print all the metadata (e.g. validity timestamps, common name or extensions) of your signed certificates. Which command did you use?
5. Check all the relevant metadata of the certificate before using it for the web server

4.2.4 Using your new certificates

Now configure nginx to use your newly signed certificate and test that it works.

1. Use the standard paths for the server certificate, server key and CA certificate. Think before you change access rights to folders or files. There are several ways to accomplish that the files can be accessed by services (service users). Use the safest one, do not use something like 0777 for files and folders. For inspiration, refer to the tasks (and their solutions) above regarding the snakeoil certificates.
2. How do you ensure that the web server can use the private key, which by default may be password protected?
3. Where do you have to put the CA certificate on the machines other than **ll-admin** so that tools like `curl` can use them?
4. Where do you have to put the CA certificate on your computer/laptop so that your browser uses it to verify the webserver certificate?
5. Check carefully that the web server presents the correct certificate and that HTTPS works:
   - Use `curl(1)` and `openssl-s_client(1)` on **ll-web**

- Use `curl(1)` and `openssl-s_client(1)` on **ll-admin**
- Use `curl(1)`, `openssl-s_client(1)` and your browser on your laptop. Which program does **not** have an option to use a socks proxy? In this case, use proxychains-ng in combination with a `proxychains.conf` file where you e.g. specify the SOCKS version, IP and Port. If you use linux, be careful to install the correct package!
  This test with proxychains is needed again for the final task and final report.
  Tip: Remember to copy the certificate of your CA to the correct path (system-wide) so that it will be used to verify the certificate signed by your CA. Alternatively, set the openssl environment variable `SSL_CERT_FILE` before running something through proxychains.

5 GNU Health

5.1 GNU Health

The goal of the following tasks is to deploy the hospital management component of the GNU Health project on multiple servers (VMs):

- Application server (Python, uWSGI)
  - Uses uWSGI, an implementation of PEP 3333 (WSGI: Web Server Gateway Interface)
  - Based on Tryton (ERP system)
  - Access via client software (GTK GUI)
- Database backend PostgreSQL RDBMS
- **Later:** Webserver (Nginx, supports the protocol uwsgi) as reverse proxy

Remember to reconfigure nftables on your VMs if new services (i.e. new ports) are being used!

The steps to full deployment of the GNU Health hospital managment system are:

1. Installation of the GNU Health application with DB backend, but **without** a reverse proxy and **without** encryption of the database connection
2. **Using** Nginx as **reverse proxy** with certificates from own CA for **HTTPS** and for **SSL/TLS encryption** of the database connection

5.2 GNU Health Deployment Part I

Start installing and configuring the database backend of the application on `ll-db`:

1. Install PostgreSQL server
2. What is the path where all configuration files are located? (some of them maybe inside subdirectories)
3. Explain all defaults in the **PostgreSQL Client Authentication Configuration File**
4. Create a new user/role ”gnuhealth”
5. Create a new database called ”health” with owner ”gnuhealth”

6. Change the **PostgreSQL Access Policy File** to allow access to the database "health" from `ll-app`

Now test the database connection from `ll-app` and install the necessary packages to deploy GNU Health with `pip` and setup/configure the application:

1. Check with `nmap` and `netcat` if the database is accessible from `ll-app`
2. Install the PostgreSQL client and connect to the database server
3. Install `python3-pip`, `python3-venv`, `libssl-dev` and `libpq-dev`
4. Create a service user "gnuhealth" with `/opt/gnuhealth` as `$HOME` and no login shell
5. Open a shell as the new "gnuhealth" user and ...
   - Create a python virtual environment (venv) in `/opt/gnuhealth/venv`, activate it and install the pip packages `gnuhealth-all-modules==4.4.1` and `uwsgi`
   - Create the folders `etc`, `var/log` and `var/lib` inside the "gnuhealth" user home directory
   - Copy the files listed in subsection 7.1 to the new `/opt/gnuhealth/etc` folder:
     - `trytond.conf`
     - `gnuhealth_log.conf`
     - `uwsgi_trytond.ini`
   - In the file `trytond.conf` ...
     - Replace `<URI>` with the correct URI
     - Set the `<DATA PATH>` to `/opt/gnuhealth/var/lib`
     - Set the file access mode to read & write for owner and group
   - Initialize the database "health" within the python venv by using `trytond-admin`. Which flags of this command are necessary to accomplish this (e.g. to use your tryton config file)? If the command works, it runs for a while and at the end appears a password prompt; what is this password for?
   - Exit the "gnuhealth" user shell
6. Use the unit file `gnuhealth.service` in subsection 7.1 to create a new systemd service (running in system mode!); see e.g. Debian Wiki
7. Enable and activate this new service, check it's status
8. Check if a GNU Health process is listening on the service port
9. Test the GNU Health application with a small python script which uses proteus, a python library to connect the the GNU Health server (which is also used by the GTK GNU Health GUI client). Install the proteus PyPI package in a venv on your `ll-admin` server and use the script in listing 2 for testing.
10. Test the application with the GNU Health client (version 4.4.0) on your laptop. Use again a python venv and install the `gnuhealth-client==4.4.0` pip package. **Before**, install all dependencies for PyGObject/GTK:
    - For Ubuntu 24.04, we recommend to follow the instructions "Installing from PyPI with pip", **but with gir1.2-gtk-3.0 instead of gir1.2-gtk-4.0**.

- For Ubuntu 22.04, libgirepository-2.0-dev is not available. So, in addition to the changes for the installation of the dependencies with apt for Ubuntu 24.04, use the package **libgirepository1.0-dev** instead. But now you have to pin the PyGObject to a version which works with this library. This means in a python venv, now do: `pip install pycairo`, then `pip install PyGObject==3.50.0` and after that install the GNU Health client with `pip install gnuhealth-client==4.4.0`.

To successfully connect to the GNU Health application on your server (`ll-app`), use ssh dynamic port forwarding (i.e. a SOCKSv5 proxy connection). Because the GNU Health client (and the proteus script if you use this also on your laptop) cannot use a SOCKSv5 proxy directly, you have to use e.g. proxychains-ng ("... a preloader which hooks calls to sockets in dynamically linked programs and redirects it through one or more socks/http proxies.").

5.3 GNU Health Deployment Part II

In this second part of the deployment of GNU Health, we use Nginx as reverse proxy and encrypt all the connections between the three parts of the application.

Tip 1: Even if you use a web server as reverse proxy, you can only test the application with our proteus script or the GNU Health client.

Tip 2: Between tests, especially if you changed something, delete the folder ~`/.config/gnuhealth/` on your laptop or wherever the GNU Health client is installed and used. The reason is that the GNU Health client might cache the wrong settings there which may cause problems.

5.3.1 Reverse proxy: HTTP

- Reconfigure your Nginx on `ll-web` to act as reverse proxy for GNU Health. Use HTTP for now to only test the reverse proxy setup without TLS.
- Adjust your proteus script `connect_to_server.py` on `ll-admin` to test if the reverse proxy works.
- Test the reverse proxy with the GNU Health client on your laptop.

5.3.2 Setup with TLS

You will use certificates, signed by your CA, for all three servers: `ll-web`, `ll-app`, and `ll-db`. The steps to set up a production-ready environment are:

1. Use TLS for connections from `ll-app` to PostgreSQL
2. Use TLS for connections from `ll-web` to GNU Health
3. Reconfigure your Nginx to use HTTPS for the connections from clients

Configure encryption for connections to your database server:

- `ll-db`: Configure the paths ot the certificate and key in `postgresql.conf`
- `ll-db`: Change the access policy configuration to only allow encrypted connections
- Test and verify that the connection is encrypted with `psql` from `ll-admin` and `ll-app`

- `ll-app`: Add query strings to the PostgreSQL URI so that the certificate will be fully verified. **Tip:** Read the chapter 34.19.1 Client Verification of Server Certificates of the PostgreSQL documentation to identify the *two* parameters you have to set in the query string. The goal is to use the configuration "[…] recommended in most security-sensitive environments".
- Test that the connection to the DB still works after setting up encryption by using the proteus script on `ll-admin` and the GNU Health client on your laptop.

Configure TLS for GNU Health:

- `ll-app`: Activate HTTPS for GNU Health and set the paths to the certificate and key in `uwsgi_trytond.ini`
  Tip: Verify that the gnuhealth user can access the certificate *and* the key. If this does not work, then check if all the directory access rights are correct.
- Test the encrypted connection with the proteus script on `ll-admin` and with the GNU Health client on your laptop.
- `ll-web`: Reconfigure Nginx so that connections from clients via HTTPS will be proxied to GNU Health. The certificate of the GNU Health server must be verified. Configure Nginx to pass the client IP to GNU Health (see e.g. Passing Request Headers, proxy_set_header, embedded variables).
- Explain all `proxy_`... and `ssl_`... settings that you use briefly; add links to sources if it is necessary or useful.
- Test the connection again with the proteus script and the GNU Health client

Some additional tests from `ll-admin` using OpenSSL:

- Test the TLS connections with `openssl` including the verification of the certificates for …
- … Nginx on `ll-web`,
- … GNU Health on `ll-app`,
- … PostgreSQL on `ll-db`. **Tip:** Because *opportunistic TLS* (what is this?) is used, an additional parameter is here needed.

6 Backup & Restore

High availability (HA) means maintaining IT services if components or subsystems fail. To achieve this, *single points of failure (SPOF)* must be avoided. Redundancy of all relevant parts (e.g. I/O paths, servers and storage) by clustering services or mirroring data is used. Typical realiability parameters are *mean time to failure (MTTF)* for a (sub)system, *mean time to recovery (MTTR)* and the sum of both times *mean time between failures (MTBF)*. These are statistical values based on experience or testing of equipment and components.

Continuous operation (CO) deals with the ability to avoid *planned* outages, such as hardware or software upgrades, maintenance of components or subsystems, and other necessary administrative work.

Continuous availability (CA) combines characteristics of HA and CO to achive the goal to reduce the likelyhood of downtimes. CA thus provides methods to ensure so-called *business continuity (BC)*.

BC includes strategies, concepts, and measures to maintain operations even in the event of an *unplanned* disruption. Two important parameters of BC are *recovery time objective (RTO)*, the targeted duration of time until a business process must be restored after a disruption. *Recovery point objective (RPO)* is the maximum acceptable interval during which data is lost. This means that the *backups* of this data must not be older than the given RPO.

6.1 Terms and concepts

Backups are regular, automated copies of data which can be used to restore (parts of) this data from a certain point in time. One can typically choose which point in time or version of a file should be restored.

There are some useful guidelines for backups, such as the *3-2-1 rule*: This means that there should be at least 3 copies of the data, stored on 2 different types of storage media, and 1 copy must be kept offside.

Several important techniques are used for *dataset optimization*. They include compression, deduplication, encryption, and hashing of the backup data.

To restrict the overall data storage size for all backups, *retention time rules* are defined which describe when old backup data will be discarded. In addition, rotation schemes of the backup media are used, like *first in, first out (FIFO)* or the more complex *Tower of Hanoi* method, based on the puzzle of the same name.

There are different backup methods: *Full backups* which include all of the data (system imaging is a special case of that), *differential backups* which include only the data change since the last full backup, and *incremental backups* which include only data changed since the last backup (full, differential or incremental). In addition, there are hybrid methods possible. This can be for example a combination of monthly full, weekly differential, and daily incremental backups. For very large and always increasing data sizes, a so-called *increment forever* method can be useful. At the beginning, one full backup will be done and afterwards only incremental backups. One disadvantage stems from the fact that the full backup, which might be several years old, and all the incremental backups until the chosen point in time for a restore are needed. This can be mitigated by calculating synthetic full backups at regular intervals and using them for restores instead of the only "real" full backup from the beginning.

6.2 (Relational) Database Backups

There are two types of backups for databases:

- *Logical backups*, which are a sequence of SQL statements to reconstruct the DB structure and data
- *Physical backups*, where the DB files are copied directly from the file system

Logical backups are independent on specific environments like DB version or system architecture. They offer more flexibility for restores than physical backups (e.g. restoring only specific database objects or tables), but may be (very) time-consuming (hours to several days). During the logical backup, changes made in the DB are not captured in the backup, which is a potential for data loss. Utilites for PostgreSQL to perform logical backups are `pg_dump`, `pg_dumpall`, and `pg_restore`.

Physical backups and especially restores can be much faster than logical ones, because here a complete "replay" of SQL operations is not required. Incremental backups are possible and *point-in-time-recovery (PITR)* is an important feature of physical backups to significantly reduce RPO. With PITR, one can restore the DB to any specific point in time between now and the time of the backup. This is especially valuable in situations where human errors or unintended changes occured. The prerequisite for PITR is an archival of the database *transaction logs*, which are a fundamental piece of most relational databases. PostgreSQL maintains a *write ahead log (WAL)* where every changes made to the database data files are recorded (see e.g. WAL Configuration and Continuous Archiving and PITR). Utilities for performing physical database backups are `(s)cp`, `rsync`, `pg_basebackup` and `Barman (Backup and Recovery Manager)`.

6.3 PostgreSQL & Barman

Barman ensures a consistent state of the DB while the server runs normally. This means that changes during the backup may be included in it. Barman offers full backups and frequent incremental backups for PostgreSQL. Important Barman concepts and the relevant parts of the PostgreSQL Backup and Restore documentation are required readings for the following tasks.

6.4 Backup & Restore of GNU Health

In this section, backups of important data will be performed and the restore tested. All backups should be stored on `ll-admin`. We use Barman to backup the PostgreSQL database. But in a GNU Health installation, not everything is stored in its database. Some files are located in `/opt/gnuhealth/var/lib` and have to be saved by other means. The file backup software restic is used to perform this task.

After the creation of the backups, the restore must be tested. We will simulate data loss by *moving* (this is safer than actually deleting) the database and test files. After the restore with Barman and restic, you have to think about how to test if everything works and perform these tests.

General questions that arise in this topic are: How do you assure the *consistency* of a backup? (Consider for example what might happen if a backup runs for a long time until it is finished.) What is needed to be able to perform a *full disaster recovery* in case the whole system (the server(s), the GNU Health installation and its data) is impacted? While we don't perform a full disaster recovery, you should describe the necessary steps in your final report.

6.4.1 Barman & GNU Health

- Install `barman` and `cron` on `ll-admin`
- What is the system user for Barman?
- Where is the home directory of the Barman system user?
- Make a copy of the barman template file for the `backup_method=postgres` in order to modify it.
- In the Barman configuration file ...
  - change the name of the config section and description in your copy;
  - change the `host` parameters to your DB server;
  - set the minimum number of backups to retain to 3
  - set the retention policy so that the maximum age of the last backup is **one month** and that only backups are kept which are needed to perform a PITR in this period
- Generate a ssh-keypair for the Barman system user
- Generate a ssh-keypair for the postgres user
- Test that both users can login as each other on the corresponding server via ssh *without a password* (these logins are used for automation, not interactively)
- Use a PostgreSQL password file to be able to connect to your database server; the first username should be `barman`, the second username `streaming_barman`

- On your DB server, create a *superuser* for the Barman connection. Then create another user for backup and WAL streaming; consult the Barman configuration file which type of PostgreSQL user this is and what privilege it needs.
- Configure two corresponding authentication records for PostgreSQL. **Tipp from the config file:** The "all" keyword does not match "replication".
- Install the package `barman-cli` on your DB server.
- Which binaries are installed with this package?
- Read their (short) man pages or the command reference in the Barman user guide to configure the `archive_command` and `restore_command` parameters of the main PostgreSQL config file.
- In the main PostgreSQL config file ...
  - enable archiving
  - check that the `wall_level` is set to `replica`
  - in the `archive_command` command, use the %p placeholder for WAL_PATH and the same for WAL_DEST in the `restore_command`
  - use the %f placeholder for WAL_NAM in the `restore_command`
  - in the `restore_command`, configure that also partial WAL files are received
- Start the barman backup job in the background if necessary: `sudo -u barman barman cron` (backup server)
- Create enough base backups to fulfill your configured minimum requirements by running `barman backup` <SERVER_NAME> as the Barman user three times
- Use Barman check to see if everything works
- Set up a cronjob to create a new base backup (running `barman backup` ... to adhear to your `last_backup_maximum_age` policy set in the Barman config
- Testing the restore: Stop the PostgreSQL service and move its data directory
  `/var/lib/postgresql/15/main`
  to simulate data loss
- Restore from the latest backup
- Check if the restore was successful in `<data_directory>/postgresql.auto.conf`
- Start PostgreSQL again and check if GNU Health is still/again working

6.5 Restic

Restic is "free to use and completely open source". It is a single executable (available for Linux, BSD, MacOS, and Windows) without dependencies (useful for restores and disaster recovery). To improve security, reproducible builds of restic are possible. Restic is secure by default (uses encrypted backups, file names; see the design document) and has a good documentation including a threat model. It offers dataset optimization like deduplication (on encrypted data), supports several storage backends, including a REST Server (self-hostable) and rclone, "the swiss army knife of cloud storage". Last but not least, it is very fast and reliable.

The files to save are located in the folder `/opt/gnuhealth/var/lib` and should be again saved on ll-admin.

- `ll-app:` Create a ssh-keypair for the gnuhealth user
- `ll-admin:` Create a new user `restic-backup` (for backup connection and storage)
- Ensure that the gnuhealth user can login a the `restic-backup` user
- Install `restic` on `ll-app` and initialize the backup repository in a folder called `restic-repo` inside the home directory of the restic-backup user
- Set the encryption password and store it in a hidden file of this home directory to be able to specify it automatically. Set the file permissions accordingly! Restic documentation
- Create a test file with known content inside the folder which will be backed up
- Create your first backup manually
- Setup a cronjob to backup every 5 minutes
- Change the content of the file and create new ones over some time to get versioned backups (restic snapshots) with different content
- List all snapshots stored in the repo
- List all files in a specific snapshot
- Check the structural consistency and integrity of your repo
- Check the integrity of the actual data that you backed up
- Move `/opt/gnuhealth/var/lib` to simulate data loss
- Restore the last snapshot with restic
- Check if all the files with their latest content are restored
- Create a new daily cronjob to remove older backup snapshots by using a policy with the following rules:
  - Keep all hourly snapshots during a day
  - Keep all daily snapshots during a week
  - Keep all weekly snapshots during a three week period
  - Keep all monthly snapshots for three months
  - Keep all yearly snapshots for two years

Computational
Health
Informatics

Leibniz
Universität
Hannover

7 Listings

7.1 Listings

Listing 2: connect_to_server.py (Replace the entries marked by angle brackets with the correct values)

```
from proteus import config
print('Connecting␣to␣the␣server...')
# First direct connection test (will be changed for later tests):
conf = config.set_xmlrpc("http://<user>:<password>@ll-app-<nr>.incus:<port>/<
    database_name>/")
# WARNING: The / at the end of the "http:// ... /" string is needed!
#          Without it, HTTP status codes like 308 or 301 occur...
print('Success')
```

Listing 3: gnuhealth_log.conf

```
[formatters]
keys: simple

[handlers]
keys: rotate, console

[loggers]
keys: root

[formatter_simple]
format: [%(asctime)s] %(levelname)s:%(name)s:%(message)s
datefmt: %a %b %d %H:%M:%S %Y

[handler_rotate]
class: handlers.TimedRotatingFileHandler
args: ('/opt/gnuhealth/var/log/gnuhealth.log', 'D', 1, 30)
formatter: simple

[handler_console]
class: StreamHandler
formatter: simple
args: (sys.stdout,)

[logger_root]
level: WARNING
handlers: rotate, console
```

Listing 4: uwsgi_trytond.ini

```
[uwsgi]
http = 0.0.0.0:8000
# https = 0.0.0.0:8443,<cert>,<key>,HIGH
wsgi-file = /opt/gnuhealth/venv/bin/trytond
env = TRYTOND_CONFIG=/opt/gnuhealth/etc/trytond.conf
env = TRYTOND_LOGGING_CONFIG=/opt/gnuhealth/etc/gnuhealth_log.conf
chdir = /opt/gnuhealth/venv
module = trytond.application:app
callable = app
stats = 127.0.0.1:9191
venv = /opt/gnuhealth/venv
logto = /opt/gnuhealth/var/log/uwsgi.log
```

Listing 5: gnuhealth.service

```
[Unit]
Description=GNU Health Server
After=syslog.target

[Service]
User=gnuhealth
ExecStart=/opt/gnuhealth/venv/bin/uwsgi --ini /opt/gnuhealth/etc/uwsgi_trytond.ini
RuntimeDirectory=uwsgi
Restart=always
KillSignal=SIGQUIT
Type=notify
NotifyAccess=all

[Install]
WantedBy=multi-user.target
```

Listing 6: trytond.conf (Replace the entries marked by angle brackets with the correct values) Because of its length, this file is also uploaded.

```
[database]
# Database related settings

# The URI to connect to the SQL database (following RFC-3986)
# uri = database://username:password@host:port/
# (Internal default: sqlite:// (i.e. a local SQLite database))
#
# PostgreSQL via Unix domain sockets
# (e.g. PostgreSQL database running on the same machine (localhost))
#uri = postgresql://tryton:tryton@/
#
# Postgres running on the same machine:
# uri = postgresql:///

# PostgreSQL via TCP/IP
# (e.g. connecting to a PostgreSQL database running on a remote machine or
# by means of md5 authentication. Needs PostgreSQL to be configured to accept
# those connections (pg_hba.conf).)
#uri = postgresql://tryton:tryton@localhost:5432/

uri = <URI>

# The path to the directory where the Tryton Server stores files.
# The server must have write permissions to this directory.
# (Internal default: /var/lib/trytond)
path = <DATA PATH>


# Shall available databases be listed in the client?
#list = True

# The number of retries of the Tryton Server when there are errors
# in a request to the database
#retry = 5

# The primary language, that is used to store entries in translatable
# fields into the database.
#language = en
```

```
[web]
listen = [::]:8000
# Settings for the web interface

# The IP/host and port number of the interface
# (Internal default: localhost:8000)
#
# Listen on all interfaces (IPv4)
#listen = 0.0.0.0:8000
#
# Listen on all interfaces (IPv4 and IPv6)

# The hostname for this interface
#hostname =

# The root path to retrieve data for GET requests
# (i.e. namely the path to the web client)
# (Internal default: /var/www/localhost/tryton)
#root = /usr/lib/node-modules/tryton-sao

# The number of proxy servers in front of trytond.
#num_proxies = 0

[webdav]
# The port on which the webdav server listens
#listen = [::]:8080

[request]
# The maximum size in bytes for unauthenticated requests (zero means no limit).
#max_size = 2MB

# The maximum size in bytes of an authenticated request (zero means no limit).
#max_size_authenticated = 2GB

[ssl]
# SSL settings
# Activation of SSL for all available protocols.
# Uncomment the following settings for key and certificate.
# SSL is activated by defining privatekey.

# The path to the private key and to the certificate
# privatekey=
# certificate=

[session]
# Session settings

# A comma separated list of login methods to use for user authentication.
# By default, Tryton supports only the password method which compares the
# password entered by the user against a stored hash.
# Other modules may define other methods (please refer to their documentation).
# The methods are tested following the order of the list.
#authentications = password

# The time (in seconds) until a session expires.
#max_age = 2592000   # (30 days)
```

```
# The time (in seconds) until an inactive session is considered invalid for
# special internal tasks, thus requiring to re-confirm the session.
#timeout = 300   # (5 minutes)

# The maximal number of authentication attempts before the server answers
# unconditionally 'Too Many Requests'.
# The counting is done on all attempts over one period of timeout.
#max_attempt = 5

# The maximal number of authentication attempts from the same network before
# the server answers unconditionally 'Too Many Requests'.
# The counting is done on all attempts over a period of timeout.
#max_attempt_ip_network = 300

# The network prefix to apply on IPv4 addresses when counting authentication attempts.
#ip_network_4 = 32

# The network prefix to apply on IPv6 addresses when counting authentication attempts.
#ip_network_6 = 56

[password]
# The minimal length required for user passwords.
#length = 8

# The path to a file containing one forbidden password per line.
#forbidden =

# The ratio of non repeated characters for user passwords.
#entropy = 0.75

# The time (in seconds) until a reset password expires.
#reset_timeout = 86400   # (24h)

# The path to the INI file to load as CryptContext:
# <https://passlib.readthedocs.io/en/stable/narr/context-tutorial.html#loading-saving-
    a-cryptcontext>
# If no path is set, Tryton will use the schemes `bcrypt` or `pbkdf2_sha512`.
#passlib = None

[email]
# Mail settings

# The URI to connect to the SMTP server.
# Available protocols are:
# - smtp: simple SMTP
# - smtp+tls: SMTP with STARTTLS
# - smtps: SMTP with SSL
#uri = smtp://localhost:25

# The From address used by the Tryton Server to send emails.
#from = tryton@localhost

[attachment]
# Defines how to store the attachments
# A boolean value to store attachment in the FileStore.
#filestore=True

# The prefix to use with the FileStore.
```

```
#store_prefix = None

[bus]
# Allow clients to subscribe to bus channels (Boolean).
#allow_subscribe = False

# The time (in seconds) to keep the connection to the client open
# when using long polling for bus messages.
#long_polling_timeout = 300

# The time (in seconds) a message should be kept in the queue
# before being discarded.
#cache_timeout = 300

# The timeout (in seconds) for the select call when listening
# on a channel.
#select_timeout = 5

# Let the worker queue handle bus messages
#queue = False

# Define the class to use when queue is set to True
#class = trytond.bus.LongPollingBus


# Special Settings
[cache]
# Various cache size settings

# The number of different models kept in the cache per transaction.
#model = 200

# The number of loaded records kept in the cache. It can also be changed
# locally using the _record_cache_size key in Transaction.context.
#record = 2000

# The number of fields to load with eager Field.loading.
#field = 100

# The minimum number of seconds between two cleanings of the cache.
#clean_timeout = 300

[queue]
# Activate asynchronous processing of the tasks. Otherwise they are performed at the
    end of the requests.
#worker = False

[table]
# This section allows to override the default generated table names. The main purpose
# is to bypass name length limitations of a database backend.
# Examples:
#account.invoice.line = acc_inv_line
#account.invoice.tax = acc_inv_tax


# Module settings
#
# Some modules are reading configuration parameters from this
# configuration file. These settings only apply when those modules
# are installed.
#
[account_fr_chorus]
# The private key to communicate with the chorus service.
#privatekey =

# The certficate to communicate with the chorus service.
#certificate =

# Target URL of the Chorus service
#url = https://chorus-pro.gouv.fr:5443

[ldap_authentication]
# The LDAP URL to connect to the server following RFC-2255.
#uri = ldap://host:port/dn?attributes?scope?filter?extensions
# A basic default URL could look like
#uri = ldap://localhost:389/

# The LDAP password used to bind if needed.
#bind_pass =

# If the LDAP server is an Active Directory.
#active_directory = False

# The UID attribute for authentication.
#uid = uid

# If the user shall be created in the database in case it does not exist.
#create_user = False

[sms_authentication]
# The fully qualified name of the method to send SMS. It must take three
# arguments: text, to and from.
#
# - The sms method just sends a code via SMS to the user. This code can directly
#   be used in the login dialog.
# - The password_sms method sends a code only after the user entered a valid
#   password (two-factor authentication).
#
# Both methods require that the user has a *mobile* phone number defined
# otherwise he can not be authenticated with those methods.
#
# This method is required to send SMS.
#function =

# The number from which the SMS are sent.
#from =

# The length of the generated code.
#length = 6

# The time to live for the generated codes in seconds.
#ttl = 300

# The name used in the SMS text.
#name = Tryton
```

```
[product]
# The number of decimals with which the unit prices are stored
# in the database. The default value is 4.
# Warning: This setting can not be lowered once a database is created.
#price_decimal = 4
```