Technology
Arts Sciences
TH Köln

DLR

Automating Multi-user Deployment for SeisComP in Linux Environment

RESEARCH REPORT

by

Subrata Roy

submitted to obtain the degree of

MASTER OF SCIENCE (M.SC.)

at

TH KÖLN UNIVERSITY OF APPLIED SCIENCES
FACULTY OF INFORMATION, MEDIA AND ELECTRICAL ENGINEERING

Course of Studies
COMMUNICATION SYSTEMS AND NETWORKS

First supervisor: Prof. Dr. Andreas Grebe
TH Köln University of Applied Sciences

Second supervisor: Dr. Stephan Sous,
MUSC, German Aerospace Center (DLR)

Cologne, September 2025

Contact details:

Subrata Roy
(Matriculation No.: 11145113)
Wetzlarer Str. 18
51105 Köln
subrata.roy@smail.th-koeln.de

Prof. Dr. Andreas Grebe
TH Köln University of Applied Sciences
Institute of Computer and Communication Technology (ICCT)
Betzdorfer Straße 2
50679 Köln
andreas.grebe@th-koeln.de

Dr. Stephan Sous
MUSC, German Aerospace Center
Linder Höhe,
51147 Köln, Germany
stephan.sous@dlr.de

Abstract

SeisComP is a widely adopted open-source software system for real-time seismic data acquisition and processing. However, its default configuration is designed for single-user environments, presenting challenges in collaborative or institutional use cases. This research project aims to design and implement a secure, scalable, and maintainable multi-user operational framework for SeisComP, using native Linux tools and automation technologies.

The project began with a manual installation of SeisComP to investigate its default behavior and identify limitations related to service control, file permissions, and user environment management. Based on the findings, a multi-user solution was designed using a dedicated system user, group-based permission handling, custom `systemd` services, and fine-grained access control through Polkit. This setup allowed multiple authorized users to manage SeisComP services without requiring root privileges.

To ensure consistency and portability, the complete solution was then automated using Ansible. Custom Ansible roles were developed to install dependencies, configure the environment, manage permissions, and deploy policy rules. The resulting system was validated through a structured series of test cases covering service accessibility, permission enforcement, and automation idempotency.

The final outcome is a robust and reusable framework that transforms SeisComP into a collaborative, secure, and automated platform suitable for institutional deployment. The approach can serve as a blueprint for similar adaptations of single-user software into multi-user environments.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my first supervisor, **Prof. Dr. Andres Grebe**, for his invaluable guidance throughout the course of this project. His clear instructions, patient explanations, and emphasis on adhering to the university's academic standards greatly helped me in structuring this report and ensuring that all necessary documents were prepared and submitted correctly. His continuous support and encouragement gave me confidence in my work, and I am truly grateful for the time and effort he invested in supervising me.

I am equally indebted to my second supervisor, **Dr. Stephan Sous**, whose insightful suggestions and constructive feedback played a significant role in shaping the quality of this work. He generously dedicated his time to reviewing my research report, scripts, providing detailed advice on both technical and written aspects, and encouraged me to refine and improve my approach at every stage. His meticulous attention to detail, combined with his constant motivation, made a remarkable difference in the development and final outcome of this project.

I would also like to acknowledge the contributions of my colleagues, **Sebastian Endres** and **Dr. Brigitte Knapmeyer-Endrun**, for their valuable collaboration and thoughtful input during the planning and deployment phases of this work. **Sebastian Endres** played a crucial role in assisting with the initial planning of the automation using Ansible and kindly reviewed parts of the code, which ensured a stronger implementation. **Dr. Brigitte Knapmeyer-Endrun**, on the other hand, supported me by providing detailed insights into the software structure, analyzing logs, and giving constructive feedback on the progress of the development. Their contributions were instrumental in refining my approach and ensuring the practical applicability of the system.

Finally, I would like to extend my heartfelt thanks to my family for their unwavering love and support throughout my academic journey. Their patience and understanding, especially during my prolonged stay abroad away from home, gave me the strength and determination to complete this work. Without their encouragement, sacrifices, and constant reassurance, this achievement would not have been possible.

Contents

List of Figures

List of Abbreviations

ACL	Access Control List
Ansible	Automation Tool for Configuration Management
Bash	Bourne Again Shell
ESA	European Space Agency
DLR	German Aerospace Center
GFZ	German Research Centre for Geosciences
GID	Group Identifier
UID	User Identifier
LDAP	Lightweight Directory Access Protocol
MUSC	Microgravity User Support Center
PID	Process ID
Polkit	PolicyKit (Linux privilege management)
SeisComP	Seismological Communication Processor
SSH	Secure Shell
UID	User Identifier
YAML	Yet Another Markup Language
`.bashrc`	User shell configuration script
`systemd`	Linux init and service manager

Chapter 1

Introduction

DLR, in Germany, stands for Deutsches Zentrum für Luft- und Raumfahrt, which means the German Aerospace Center [1]. It is the national research center for aeronautics and space, as well as energy, transport, and security. The Microgravity User Support Center (MUSC) at the German Aerospace Center (DLR) in Cologne is a facility that provides scientific and technical infrastructure for conducting experiments in microgravity. MUSC supports European researchers in various fields, including bio and material science under microgravity conditions [2]. The LUNA facility at the DLR Cologne is an analogue lunar simulation facility, jointly operated by the German Aerospace Center (DLR) and the European Space Agency (ESA). It aims to replicate conditions on the lunar surface to prepare astronauts and robots for future missions to the Moon. The facility features a 700 square meter regolith area, a deep floor area, a dust chamber, and other specialized areas for testing and development [3].

In this facility, along with other sensors, a seismometer was installed to collect and analyze the data measured from different experiments. For this, SeisComP software is required to be installed on servers at the far site that will collect the data from the centaur and analyze it further for scientific research. SeisComP is a powerful open-source software package designed for real-time seismic data acquisition, processing, monitoring, and archiving. It is widely used by research institutions and seismic networks worldwide for earthquake detection and analysis [4].

1.1 Problem Statement

With the regular installation of SeisComp Software, only the user who installed and started the service can operate the service. This is a challenge in a collaborative environment, where multiple users are involved in managing the SeisComP operation.

- **Single-User Limitation:** Only one user can operate this service.
- **File Permission:** Generated files are owned by the user who operates the service and hence other users cannot view the service status.
- **Manual Installation:** Lack of automation increases deployment time and introduces human errors.

In its default configuration, SeisComP is designed to operate in a single-user environment, which restricts system administration to the user who performed the installation. This creates significant challenges in production environments where multiple users need to interact with or manage the services. Issues arise related to file ownership, permission conflicts, service visibility, and secure user management.

Without a centralized or group-based privilege model, users outside the original installing account cannot effectively start, stop, or monitor SeisComP services, leading to workflow disruptions and administrative bottlenecks.

1.2 Goals, Requirements, and Scope of work

The primary goal of this project is to build a robust, secure, and scalable multi-user environment for SeisComP. To achieve this, the following objectives were defined:

- Analyze the limitations of the default single-user setup.
- Design a system that enables selected users or groups to manage SeisComP services without compromising security.
- Develop a solution using native Linux tools such as systemd, user groups, and permissions.
- Implement automation using Ansible to ensure consistent deployment and easy scalability.

The scope of this project is limited to Linux-based environments, particularly Debian systems, and focuses on backend operational improvements. Frontend application changes and integration with external monitoring systems are not within the scope.

1.3 Methodology

This project employed a progressive, problem-driven methodology. The initial phase involved installing SeisComP manually to examine its default behavior in a single-user environment. During this phase, several limitations were observed and analyzed.

Each identified issue was addressed using native Linux mechanisms. This process led to the creation of a working multi-user configuration, supported by custom services and permission management.

After validating the manual setup, the solution was automated using Ansible roles. This ensured that future deployments would be reusable, consistent, and efficient.

The specific implementation steps—including configuration details and automation logic—are described in the following chapters.

1.4 Conclusion

This chapter presented the background and motivation for the project, highlighting the limitations of single-user SeisComP deployments. It outlined the project objectives and the adopted methodology. The subsequent chapters will explore the relevant technologies, describe the step-by-step manual implementation, address the associated challenges, and conclude with the fully automated solution using Ansible.

Chapter 2

Technical Background

2.1 Introduction

This chapter presents the technical background of the key tools, services, and concepts used throughout the research project. These technologies form the foundation of the implementation steps described in later chapters. Topics include the architecture and operational flow of SeisComP, user and permission management in Linux, service handling with systemd, scripting with Bash, privilege management with Polkit, user environment configuration, and automation using Ansible.

2.2 SeisComP Overview

SeisComP was originally developed by the GFZ German Research Centre for Geosciences and now jointly maintained by gempa GmbH. It has evolved over nearly two decades into a modular, flexible, and community-supported platform. SeisComP is used extensively by seismic monitoring agencies, research centers, and observatories around the world. With its blend of automatic and interactive capabilities, it supports comprehensive workflows ranging from real-time event detection to manual event review [4].

At the core of SeisComP's real-time data exchange is the SeedLink protocol. It is a highly efficient and reliable streaming protocol that enables the continuous transmission of seismic waveform data from remote sensors to central processing systems. It is based on TCP, and the client initiates the connection. Due to its robustness and simplicity, SeedLink has become a de facto standard in the seismological community [5].

SeisComP follows a modular architecture, where each component is designed to perform a specific task within the broader seismic monitoring workflow [6]. This loosely coupled design not only promotes scalability but also enhances system robustness and fault tolerance. While the system includes a wide range of modules, this section offers a brief overview of only a few key components relevant to the context of this research. The primary focus here is not an exhaustive exploration of SeisComP itself, but rather the administration and multi-user configuration of the software within a Linux environment.

- **scmaster**: Acts as the central coordinator for all other modules, managing event communication, station data flow, and the overall internal messaging architecture [7].

- **seedlink**: A data acquisition module that receives real-time waveform data from seismic sensors and streams it to clients. It is the entry point of live data into the SeisComP system [5].
- **slarchive**: Connects to a SeedLink server and archives incoming waveform data into standard MiniSEED files for long-term storage and later analysis [8].
- **scheli**: Provides a helicorder-style visualization of seismic data, allowing users to visually monitor waveforms across time in a familiar format [9].
- **scrttv**: A graphical interface used for real-time waveform inspection and manual phase picking. It is particularly useful in monitoring scenarios and during manual quality control [10].

These components work collaboratively to ensure that seismic data is not only captured and stored efficiently but is also accessible and reviewable in both automated and interactive ways, making SeisComP highly effective for real-time earthquake monitoring and post-event analysis.

2.3 Linux File System, Users, and Permissions

Linux is a multi-user operating system that provides robust mechanisms for managing files, directories, and access permissions. These mechanisms are essential for maintaining system security, ensuring user isolation, and enabling controlled collaboration. In the context of deploying and managing software like SeisComP in a shared environment, understanding how Linux handles file ownership and access control is fundamental. This section provides a brief overview of key concepts such as the Linux file hierarchy, user and group structures, and permission models, with particular emphasis on their relevance to system administration tasks.

Linux organizes data in a hierarchical file system rooted at `/`. Key directories include `/etc/` for configuration files, `/home` for user home directories, `/var` for variable data such as logs, and `/usr` for user programs and libraries. Files or directories in Unix-like systems are pointed to an inode. An inode is a data structure that stores all metadata associated with a filesystem object, including ownership and permissions. When a process attempts to access a file, the kernel checks this metadata and compares it with the process's credentials to determine the appropriate access rights [11].

System access is managed through users and groups. Each user, identified by a unique user ID (UID), is either the superuser (root) with unrestricted privileges or a regular user with limited access. Users are organized into groups, identified by group IDs (GIDs), to facilitate shared access to resources. User and group information is stored in `/etc/passwd` and `/etc/group`, respectively [12]. This structure enables efficient management of permissions across multiple users.

Permissions in Linux are categorized as read (r), write (w), and execute (x), applied to three classes: the file's owner, the group, and others. Permissions are represented symbolically (e.g., `rwxr-xr-x`) or numerically in octal form (e.g., 755, where 7 = rwx, 5 = r-x). Commands such as `chmod` modify permissions, while `chown` and `chgrp` manage ownership, and `umask` are essential in adjusting these permissions. For advanced use cases, access control lists (ACLs) provide finer-grained control, allowing specific permissions for individual users or groups beyond the standard model [13], [14].

These mechanisms collectively ensure secure and controlled access to system resources, forming the backbone of Linux system administration. Understanding these components is essential for analyzing application behavior in implementing secure multi-access SeisComP in this research project.

2.4 Systemd Services and Custom Unit Configuration

Modern Linux distributions adopt `systemd` as the standard init system and service manager. It replaces the traditional SysV init system and provides an efficient framework for managing services, daemons, and other system processes [15]. At its core, `systemd` uses *unit files* to define the behavior and lifecycle of services, offering administrators precise control over startup order, user privileges, restart policies, and execution environments [16].

One key advantage of `systemd` is its declarative configuration approach. Rather than relying on ad hoc scripts, system services are defined in structured files with the `.service` extension, stored typically in `/etc/systemd/system/` for custom configurations. These services can be easily managed using the `systemctl` command-line utility, which facilitates starting, stopping, enabling, disabling, and inspecting services [17], [16].

Creating a Custom `systemd` Service

Creating and managing a custom service in Linux involves the following general steps:

1. Create a new unit file, e.g., `/etc/systemd/system/example.service`.
2. Define the required sections:
   - **[Unit]**: Describes metadata and dependencies.
   - **[Service]**: Contains execution details like commands, user, group, and environment.
   - **[Install]**: Defines how the service integrates into system targets (e.g., `multi-user.target`).
3. Reload `systemd` to acknowledge the new service:
   ```
   sudo systemctl daemon-reload
   ```
4. Enable and start the service:
   ```
   sudo systemctl enable example.service
   sudo systemctl start example.service
   ```

Custom services provide flexibility in managing software execution, enabling administrators to specify permissions, runtime behavior, and even post-start tasks. In this project, custom systemd services were created, which are detailed in Chapter 3, Section 3.3.2

2.5 Linux Bash Scripting

The Bourne Again Shell (Bash) serves as both a widely used command interpreter and a full-featured scripting language in Linux environments. Its scripting capabilities allow users to write reusable logic in '.sh' files by combining command-line utilities with control structures such as loops and conditionals [18], [19]. Bash is particularly effective for automating routine tasks, chaining multiple commands, and integrating with other Linux tools, such as `grep`, `awk`, and `sed`.

In research and system administration contexts, Bash scripting enhances consistency, efficiency, and reproducibility. It can be used to manage services, enforce permission logic, or coordinate software operations in an automated and repeatable manner. Within this research project, Bash scripting played a key role in automating service-related operations.

2.6 Polkit: Controlled Privilege Escalation

Modern Linux systems often require a mechanism to securely delegate limited administrative privileges to non-privileged users without granting full root access. **Polkit** (formerly known as PolicyKit) serves this purpose by providing a flexible framework for defining and enforcing access policies for privileged operations.

Polkit operates between unprivileged user processes and privileged system components. When a user attempts an action that requires elevated permissions—such as starting or stopping a system service—Polkit evaluates whether the request is authorized according to its configured policies. These policies can grant or deny access based on user identity, group membership, the action requested, or interactive authentication [20],[21].

Unlike traditional approaches such as adding users to the `sudoers` file, Polkit allows for fine-grained privilege control without handing out blanket administrative rights. It does this by using JavaScript-based rule files (typically located under `/etc/polkit-1/rules.d/` or `/usr/share/polkit-1/rules.d/`), which define logic for what actions are permitted for which users or groups [22],[23].

In practice, Polkit is often used alongside `systemd` to control access to service management commands like `systemctl start` or `systemctl restart`. This approach enables non-root users—who belong to a specific group—to perform specific actions (e.g., restarting a custom service) without requiring full system privileges.

In the context of this research project, Polkit plays a key role in enabling users within a designated group to manage the SeisComP service securely, maintaining operational control while preserving system security boundaries.

2.7 Managing the User Environment

In a Linux system, the **user environment** refers to the collection of settings and configurations that define how the system behaves for a specific user upon logging in to the system. These settings influence aspects such as command execution, environment variables, and access to system resources, playing a vital role in shaping the user's interaction with the shell and various applications.

A key component of the user environment is the `~/.bashrc` file, which is executed whenever a user starts a new interactive shell session. It allows users or administrators to customize the shell by setting environment variables (such as `PATH`), defining aliases for frequently used commands, or enabling context-specific behaviors. These customizations enhance efficiency and provide a more consistent and tailored user experience.

From a system administration perspective, customizing the user environment plays an important role in tailoring the behavior of applications based on roles or group membership. For example, loading specific scripts, setting paths to binaries, or enabling tab completion for domain-specific tools can improve both usability and consistency across users. These environment configurations are modified globally using files like those under `/etc/profile.d/` directory [24].

Customizing the user environment was essential for enabling seamless access to SeisComP commands and configurations in this project. By loading application-specific configurations automatically for authorized users or groups, the system ensures both usability and security.

2.8 Ansible for Automation

Ansible is a powerful open-source automation tool widely used for configuration management, application deployment, and orchestration of system tasks. Designed with simplicity and scalability in mind, Ansible utilizes SSH to manage remote systems without requiring additional software (agents) on the managed hosts. Its declarative, human-readable YAML syntax and modular design make it particularly suitable for both small and complex infrastructure management [25], [26].

2.8.1 Key Features and Benefits

The following key benefits of Ansible have been summarized from the official documentation and related sources [25], [26],[27]:

- **Agentless Architecture:** Ansible communicates over SSH, eliminating the need to install any agents on target systems.
- **Simple and Readable Syntax:** Playbooks are written in YAML, which is both easy to write and understand.
- **Idempotency:** Tasks are designed to be idempotent, meaning they can be safely repeated without changing the result unless a change is required.
- **Cross-Platform Support:** Ansible works on various Unix-like systems and integrates well with cloud platforms.
- **Scalable:** It can automate operations across many nodes simultaneously, suitable for both personal and enterprise-scale environments.

2.8.2 Ansible Project Directory Structure Overview

The Ansible official documentation has provided detailed description regarding the directory structure [28]. A typical ansible project is organized into a well-defined directory structure that promotes modularity and maintainability:

```
.
ansible-project/
 inventory/
    hosts.ini
 playbooks/
    site.yml
 roles/
    apache/
       tasks/
       templates/
 group_vars/
    all.yml
 host_vars/
    all.yml
 ansible.cfg
```

Each directory serves a specific purpose. The `inventory` defines managed hosts, `playbooks` contain the automation logic, `roles` structure reusable components, and `group_vars` or `host_vars` store variables per group or per host respectively.

2.8.3 Playbooks and Tasks

Playbooks are YAML files that define sets of tasks to be executed on one or more remote systems [29]. Each play targets a group of hosts and defines the execution logic:

```yaml
- name: Install Apache
  hosts: webservers
  become: yes
  tasks:
    - name: Install Apache
      ansible.builtin.apt:
        name: apache2
        state: present
```

FIGURE 2.1: Basic Ansible Playbook Example

2.8.4 Ansible Modules

Modules are the core components of Ansible's task execution engine. They perform discrete functions such as installing packages, copying files, managing services, or handling users [30], [31]. Below are some commonly used modules:

- `ansible.builtin.apt` – Manages packages on Debian/Ubuntu systems [32].
- `ansible.builtin.copy` – Copies files to target nodes [33].
- `ansible.builtin.template` – Deploys Jinja2 templates [34].
- `ansible.builtin.systemd` – Manages `systemd` services [35].
- `ansible.builtin.user` – Creates or modifies user accounts [36]

These modules are declarative, meaning they describe the desired state of the system, and Ansible ensures the system matches that state.

2.8.5 Ansible Roles

Roles in Ansible are a standardized method of organizing playbooks and reusable automation logic. According to ansible official documentation [37], a role might contain at least one directory among seven standardized directories as below:

```
roles/
    common/               # this hierarchy represents a "role"
        tasks/            #
            main.yml      #  <-- tasks file can include smaller files if warranted
        handlers/         #
            main.yml      #  <-- handlers file
        templates/        #  <-- files for use with the template resource
            ntp.conf.j2   #  <------- templates end in .j2
        files/            #
            bar.txt       #  <-- files for use with the copy resource
            foo.sh        #  <-- script files for use with the script resource
        vars/             #
            main.yml      #  <-- variables associated with this role
        defaults/         #
            main.yml      #  <-- default lower priority variables for this role
        meta/             #
            main.yml      #  <-- role dependencies
        library/          # roles can also include custom modules
        module_utils/     # roles can also include custom module_utils
        lookup_plugins/   # or other types of plugins, like lookup in this case

    webtier/              # same kind of structure as "common" was above, done for the webtier role
    monitoring/           # ""
    fooapp/               # ""

```

FIGURE 2.2: Ansible Role structure Example [37]

Roles enhance readability, facilitate reuse across projects, and make it easier to share automation logic within teams.

2.8.6 Secure Automation with Ansible Vault

Ansible Vault provides a mechanism to encrypt sensitive data such as passwords or private keys. This is crucial in maintaining clean, secure playbooks without exposing credentials [38].

Examples of vault operations include:

- Create a new vault file: `ansible-vault create secrets.yml`
- Edit a vault file: `ansible-vault edit secrets.yml`
- Encrypt an existing file: `ansible-vault encrypt secrets.yml`

Vault-encrypted files are typically referenced in `group_vars` or `host_vars` and are decrypted during execution using a password prompt or a vault ID.

2.9 Conclusion

This chapter introduced the key technologies that form the foundation of this project, including SeisComP, Linux file and permission management, `systemd` services, Bash scripting, `polkit`, and user environment configuration. It also covered automation with Ansible, highlighting its playbooks, roles, modules, and secure practices using Ansible Vault. These concepts provide the technical groundwork for the multi-user automation strategies discussed in the following chapters.

Chapter 3

Solution Development and Implementation

3.1 Introduction

This chapter presents the solution development and implementation details of the project, outlining the step-by-step approach taken to transform the default single-user SeisComP setup into a secure and maintainable multi-user environment. The work was carried out in three phases: manual installation and evaluation of SeisComP, implementation of a group-based manual solution to overcome user limitations, and automation of the entire setup using Ansible roles.

3.2 Manual Installation and Problem Identification

The project began with the standard manual installation of SeisComP on a Debian-based system. The software was extracted from a tarball package, and initial configurations were performed under a single system user as described in appendices (A.1). The following key observations were made during the multi-user testing of the SeisComP service with this manual installation (A.2):

- Only the user who initiates the SeisComP service can start, stop, or check its status.
- Users from the same group, despite having identical group permissions, are unable to interact with the service.
- Temporary files such as process IDs (PIDs) and logs are generated under directories like `/seiscomp/var/run`, and are owned by the user who starts the service.
- File ownership restrictions prevent other users from accessing or managing service-related files, limiting multi-user collaboration.

To address this, the next section explores a manual solution leveraging native Linux mechanisms to enable controlled multi-user access — a foundational step toward eventual automation using Ansible.

3.3 Manual Multi-User Solution

To overcome the challenges identified in the previous section, a series of manual configurations were applied using native Linux tools and mechanisms. The objective was to enable secure and collaborative multi-user management of SeisComP services without compromising system integrity. This section outlines the step-by-step solutions, including user and group management, service permission handling through custom systemd units, privilege delegation using Polkit, and user environment adjustments to streamline access and usability.

3.3.1 System User and Group Configuration

A dedicated system user was created to run SeisComP services. A new user group (e.g., `sysop`) was created, and all authorized users were added to this group. The ownership of the SeisComP directory structure and data files was adjusted to `seiscomp:sysop`, with group read/write permissions enabled.

3.3.2 Custom Systemd Service

Managing service ownership and access in a multi-user environment requires precise control over how system processes are initiated and how they interact with the file system. In this context, two custom `systemd` service units were introduced to address the key limitations identified earlier. The first ensures that any new files or directories generated within the SeisComP environment are assigned the correct ownership and permissions. The second guarantees that the SeisComP service is always executed under a dedicated system user, regardless of which user initiates the command. These services work in tandem to enable a consistent, secure, and collaborative runtime setup for all authorized users.

Fix SeisComP Permission

During the initial manual installation, it was observed that executing the `seiscomp start` command resulted in the creation of temporary files and process-related directories (e.g., under `/opt/seiscomp/var/run`) that were owned by the user who started the service. These files did not inherit the intended group permissions, thereby restricting access for other authorized users within the same group. To address this, a solution was devised to monitor changes within the `/opt/seiscomp` directory and automatically enforce the correct ownership and permission settings. A custom Bash script (Figure 3.1) was developed to watch for new or modified files and recursively update their ownership to `seiscomp:sysop`, while also aligning group permissions with user permissions using `chmod g=u`. Additionally, it also writes change logs to the file `/var/log/fix_seiscomp_permission.log`.

```bash
#!/bin/bash

WATCH_DIRS=(
    "/opt/seiscomp/"
)

LOGFILE="/var/log/fix_seiscomp_permission.log"

# Function to add inotify watch to a directory
add_watch() {
    local DIR="$1"
    echo "$(date): Starting recursive monitoring on $DIR" >> "$LOGFILE"
    inotifywait -m -r -e create,modify,move "$DIR" --format "%w%f" |
    while read -r FILE; do
        if [ -e "$FILE" ]; then
            chmod g=u "$FILE"
            echo "$(date): Changed permission for $FILE" >> "$LOGFILE"
        fi
        sleep 2
    done
}

# Call the function to watch the directory
add_watch "${WATCH_DIRS[@]}"
```

FIGURE 3.1: Fix SeisComP Permission Bash Script

As outlined in Chapter 2, Section 2.4, systemd provides a robust mechanism for managing persistent background services. Following this approach, a custom systemd unit service was created to ensure the continuous and reliable execution of the permission fixing script. The Bash script (Figure 3.1) was stored in `/usr/local/bin`, and the unit file was placed in `/etc/systemd/system/fix_seiscomp_permission.service` (Figure 3.2). To support real-time file system monitoring, the `inotify-tools` package was installed as a required dependency.

```ini
[Unit]
Description=Fix Permission Service
After=network.target

[Service]
Type=simple

ExecStart=/usr/local/bin/fix_seiscomp_permission.sh
Restart=always
User=seiscomp
Group= sysop

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

FIGURE 3.2: Code Snippets: Fix SeisComP Permission service

SeisComP Service Wrapper

In its default configuration, the `seiscomp` command-line utility internally executes scripts such as `/opt/seiscomp/bin/seiscomp start`, `stop`, or `restart`. When SeisComP is configured system-wide, an environment setup file (e.g., `/etc/profile.d/seiscomp.sh`) ensures that these commands are accessible from any terminal session, effectively mapping the `seiscomp` command to its binary location. However, this default behavior means that the service runs under the identity of the user who invokes the command. Consequently, the generated runtime files and processes are owned by that user, reintroducing the ownership and access control issues previously discussed.

To standardize service execution and maintain consistent ownership, a second custom `systemd` unit file—`seiscomp.service`(Figure 3.3) was created. This wrapper service ensures that the SeisComP process is always launched as the dedicated `seiscomp` system user, regardless of who initiates the service control action. By enforcing a single execution identity, this design promotes security, simplifies permission management, and enables seamless collaboration among users within the authorized group.

```
[Unit]
Description=SeisComp Service
After=network.target

[Service]
Type=forking

User=seiscomp
Group=sysop

WorkingDirectory=/opt/seiscomp

# File Permission settings before start the service
ExecStartPre=/bin/chown -R seiscomp:sysop /opt/seiscomp
ExecStartPre=/bin/chmod -R g=u /opt/seiscomp

# Command Execution
ExecStart=/opt/seiscomp/bin/seiscomp start
ExecStop=/opt/seiscomp/bin/seiscomp stop
ExecReload=/opt/seiscomp/bin/seiscomp restart

# File permission settings
UMask=0002
PermissionsStartOnly=true

# File Permission after the service start
ExecStartPost=/bin/chown -R seiscomp:sysop /opt/seiscomp
ExecStartPost=/bin/chmod -R g=u /opt/seiscomp

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

FIGURE 3.3: Code Snippets: SeisComP service

3.3.3 Polkit Integration for Service Control

While the custom `systemd` services introduced in the previous section enable standardized management of SeisComP processes, invoking these services typically requires elevated privileges. Non-sudo users are not permitted to execute commands such as `systemctl start seiscomp.service` by default, as `systemd` operations are restricted to users with administrative access.

To address this limitation, and as introduced in Chapter 2, Section 2.6, `polkit` was employed to delegate limited administrative capabilities to authorized users in a controlled and secure manner. Specifically, a custom policy rule was defined to allow members of the `sysop` group to manage the `seiscomp.service` without full sudo privileges. This rule grants the ability to start, stop, and restart the service while preserving system-level protections and minimizing the risk of unauthorized access to other privileged operations.

```
polkit.addRule(function(action, subject) {
  if (
    action.id == "org.freedesktop.systemd1.manage-units" &&
    action.lookup("unit") == "seiscomp.service" &&
    subject.isInGroup("sysop")
  ) {
    return polkit.Result.YES;
  }
});
```

FIGURE 3.4: Polkit rule for non sudo users group

The rule was implemented in the file `/etc/polkit-1/rules.d/50-seiscomp.rules`, as shown in Figure 3.4, and it safely delegates control of the service to users in the specified group "sysop".

3.3.4 User Environment Management

To ensure a seamless experience for authorized users, environment variables and command behavior were configured through system-wide shell initialization scripts. By default, SeisComP requires several environment variables—such as `SEISCOMP_ROOT`, `PATH`, and `LD_LIBRARY_PATH`—to be properly set before its commands can be executed. These were centrally defined in `/etc/profile.d/seiscomp.sh` (Figure 3.5), which ensures they are applied automatically to all interactive shells.

```
export SEISCOMP_ROOT="/opt/seiscomp"
export PATH="/opt/seiscomp/bin:$PATH"
export LD_LIBRARY_PATH="/opt/seiscomp/lib:$LD_LIBRARY_PATH"
export PYTHONPATH="/opt/seiscomp/lib/python:$PYTHONPATH"
export MANPATH="/opt/seiscomp/share/man:$MANPATH"
source "/opt/seiscomp/share/shell-completion/seiscomp.bash"
```

FIGURE 3.5: Default SeisComP environment configuration

However, due to the custom systemd "seiscomp service" configuration (see Section 3.3.2), users are required to interact with the SeisComP service using `systemctl` commands rather than the traditional `seiscomp` CLI tool. To maintain consistency and improve usability, a secondary environment script-`"/etc/profile.d/luna.seiscomp.sh"` was introduced (Figure 3.6). This custom script performs two key functions: it restricts its scope to users in the `sysop` group, and it overrides the default `seiscomp` command for selected operations. Specifically, commands such as `seiscomp start`, `stop`, and `restart` are redirected to the corresponding `systemctl` actions for the `seiscomp.service` unit. In contrast, status-related commands like `seiscomp status` continue to invoke the native executable directly, as group-based permissions allow access to runtime information.

```bash
#!/usr/bin/env bash

# Only apply to users in 'sysop' group
if id -nG "$USER" | grep -qw sysop; then

  # Source the main SeisComP environment
  [ -f /etc/profile.d/seiscomp.sh ] && source /etc/profile.d/seiscomp.sh

  # Override seiscomp command using systemctl
  seiscomp() {
    case "$1" in
      restart) systemctl restart seiscomp.service     ;;
      start)   systemctl start seiscomp.service       ;;
      stop)    systemctl stop seiscomp.service        ;;
      *)       /opt/seiscomp/bin/seiscomp "$@"        ;;
    esac
  }
  export -f seiscomp
fi
```

FIGURE 3.6: Custom SeisComP command override for sysop group

This approach ensures that users can continue to interact with SeisComP using familiar commands, while preserving the integrity and control provided by the underlying custom service and permission model. This script ensures all group members have access to SeisComP commands upon login.

3.3.5 Manual Solution: Multi-user SeisComP in a Nutshell

The manual configuration for enabling multi-user access to SeisComP services integrates several Linux-native components working together to ensure a secure and collaborative environment. As illustrated in Figure 3.7, the process begins when a user logs into the system.

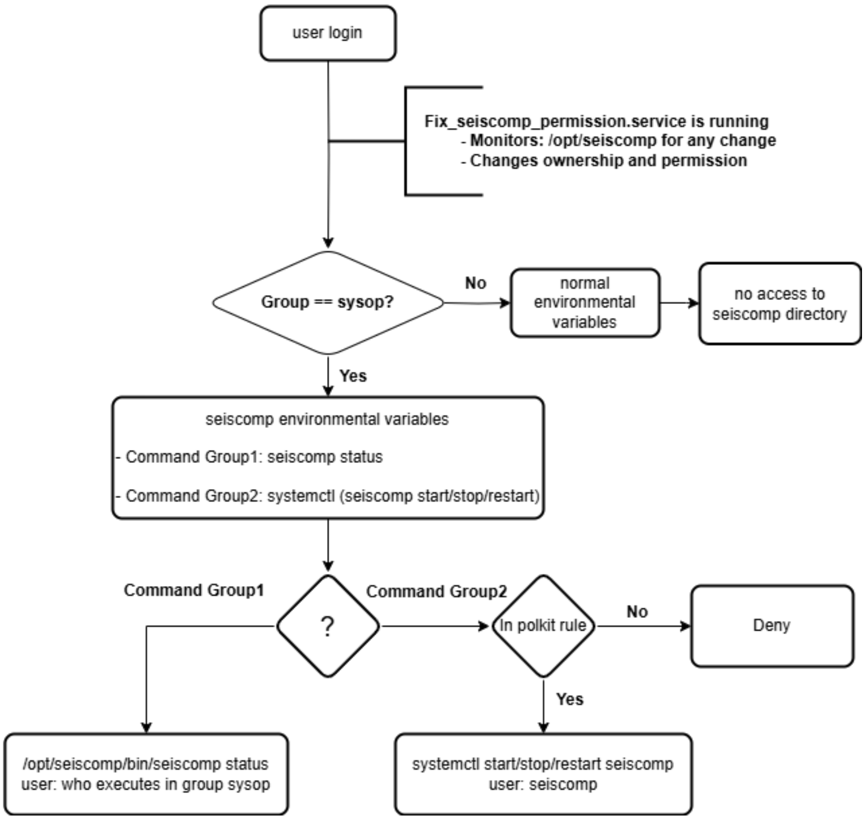


FIGURE 3.7: Multi-user SeisComP deployment workflow

If the user is not part of the `sysop` group, only standard environment variables are loaded, and access to the SeisComP directory is denied. For users in the `sysop` group, the system automatically loads SeisComP-specific environment variables, which define executable paths and library configurations. These users' commands are granted two levels of interaction: one command group is permitted to query the status of the SeisComP service using the default binary, while another command group (subject to a polkit rule) is authorized to control the service using `systemctl` without requiring root privileges. Meanwhile, the `fix_seiscomp_permission.service` continuously monitors the SeisComP directory for changes, ensuring that all generated files maintain the correct ownership and permission scheme. Together, these mechanisms establish a functional multi-user runtime model for managing SeisComP.

3.4 Automation with Ansible Roles

Once the manual setup was validated in section 3.3, it was translated into a fully automated Ansible role. This improved reproducibility and allowed easy deployment across multiple systems. The automation was broken down into two main roles: one for provisioning the MariaDB server, and another for configuring SeisComP with all the supporting components required for multi-user operation.

3.4.1 Ansible MariaDB Role

MariaDB is a prerequisite for SeisComP when it is selected as the application's database backend. Rather than embedding database configuration within the main SeisComP role, MariaDB was implemented as an independent role to take advantage of the modular design principles outlined in Section 2.8.5. This separation allows the MariaDB role to be reused in other projects or services that require a similar database setup.

The role provisions the MariaDB server, performs the initial security hardening according to the official documentation [39], and sets the root password. It also ensures that the database service is active and configured to start at boot. By handling these tasks in a dedicated role, the database can be provisioned independently or alongside SeisComP, improving flexibility and maintainability. The full task sequence for the MariaDB role, including YAML definitions and handlers, is provided in Appendix A.3.6.

3.4.2 Ansible SeisComP Role

The SeisComP Ansible role encapsulates all tasks required to replicate the manual multi-user configuration described in Section 3.3, but in a fully automated and repeatable form. Unlike the MariaDB role, which is dedicated solely to provisioning and securing the database service, the SeisComP role incorporates an initial check to verify whether a functional MariaDB instance is already present. If the database is not available, the role triggers the MariaDB installation automatically, ensuring that the necessary prerequisite is in place. This design makes the role more flexible and self-contained, while still aligning with the modular approach of Ansible roles. Once the database dependency is satisfied, the role proceeds with the complete workflow for deploying and configuring SeisComP in a multi-user environment.

Within this role, tasks are organized to address the entire multi-user deployment workflow:

1. **User and Group Provisioning** – Creates the dedicated system user `seiscomp` and the administrative group `sysop`, adding all authorized operators to this group.
2. **Directory Preparation** – Creates `/opt/seiscomp` with recursive ownership set to `seiscomp:sysop` and permissions aligned (`g=u`) for consistent group access.
3. **Software Installation** – Downloads, extracts, and installs SeisComP, along with all required dependencies (including `seiscomp-server`).
4. **Database Integration** – Configures SeisComP to connect to the MariaDB instance prepared earlier.
5. **Environment Setup** – Deploys the standard SeisComP environment script to `/etc/profile.d` and installs a conditional wrapper script to simplify service management for `sysop` members.
6. **Custom Systemd Services** – Installs and enables two custom unit files:
   - `fix_seiscomp_permission.service` – Monitors `/opt/seiscomp` for newly created files or directories and corrects ownership and permissions in real time.
   - `seiscomp.service` – Ensures that SeisComP processes always run under the `seiscomp` user, regardless of the initiating account.
7. **Polkit Integration** – Configures a rule to allow `sysop` members to control `seiscomp.service` without requiring `sudo` privileges.
8. **Post-Deployment** – Enable modules, and restart the services.

The complete breakdown of the SeisComP role, including the YAML playbooks, templates, and handlers, etc, is provided in Appendix A.3.7.

3.4.3 Summary

By implementing SeisComP deployment as an Ansible role and separating MariaDB into its own reusable module, the solution achieves both *reliability* and *scalability*. Any system meeting the base requirements can now be configured for multi-user SeisComP operation with a single playbook run, eliminating the manual intervention previously required. This approach not only reduces setup time but also ensures consistency across environments, aligning with the principles of infrastructure as code.

Chapter 4

Testing and Evaluation

4.1 Introduction

This chapter presents the testing procedures and evaluation results for the multi-user SeisComP environment developed during the project. The aim of the testing phase was to validate the manual and automated configurations, ensure expected behaviour under multiple user accounts, and confirm that the deployment process is reproducible using Ansible. The evaluation focuses on functional verification, permission and access control, service reliability, and automation consistency.

4.2 Testing Environment

The testing was conducted in a controlled virtualised environment using VMware Workstation 17. The setup comprised three virtual machines (VMs):

- **Workstation:** An Ansible control node configured with the IP address `192.168.101.250/24`.
- **Servers:** Two Debian 12 VMs (`192.168.101.201` and `192.168.101.202`) designated as deployment targets for the SeisComP application via Ansible roles.

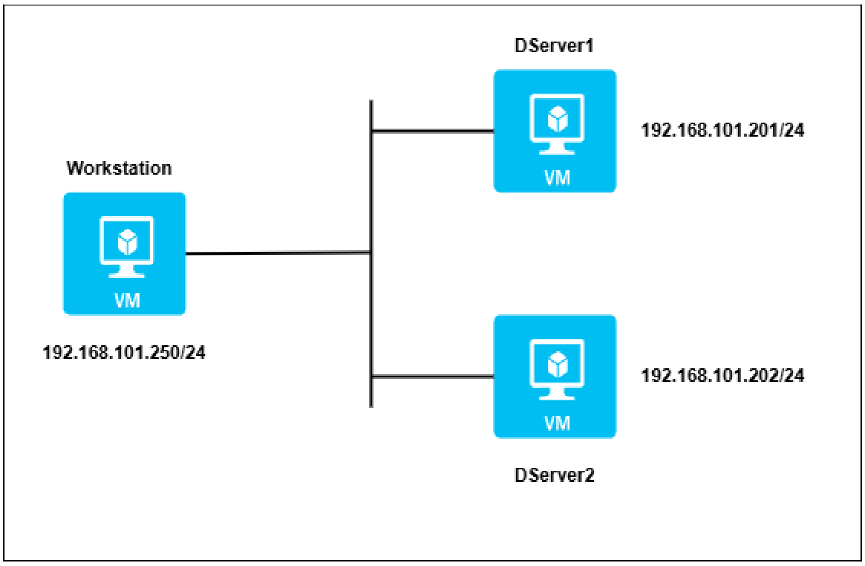


FIGURE 4.1: LAB Test Diagram

A simplified network topology diagram is shown in Figure 4.1, illustrating the logical connectivity between these systems. Each VM was equipped with two network interfaces:

1. A NAT interface for internet access (not shown in the diagram).
2. A management interface connected to the management network `192.168.101.0/24`.

4.2.1 Ansible Environment Preparation

The Ansible control node was configured to manage the two Debian 12 servers using passwordless SSH authentication, an inventory configuration, and project-specific Ansible settings. Sensitive credentials such as database passwords were securely managed using Ansible Vault, as described in Section 2.8.6.
The full preparation process, including package installation, key-based authentication, vault encryption, and configuration files, is detailed in Appendix A.3.1.

4.2.2 Executing the Ansible Playbook

As outlined in Section 2.8.3, a dedicated playbook named `pb_install_service.yml` was created to orchestrate the deployment of both the `mariadb` and `seiscomp` roles. The playbook was executed in two stages to isolate role-specific installations and verify their outcomes individually:

```
ansible-playbook pb_install_service.yml --limit dev_servers --tags
mariadb --ask-vault-pass
ansible-playbook pb_install_service.yml --limit dev_servers --tags
seiscomp --ask-vault-pass
```

This approach allowed targeted testing of each component—first provisioning the MariaDB service, followed by the installation and configuration of SeisComP. The complete playbook listing is available in Appendix A.4.

4.3 Test Cases and Objectives

The following test cases were designed to cover the most critical aspects of the system:

4.3.1 Test Case 1: Service Control by Group Members and Polkit Authorization Validation

Objective: Verify that users in the `sysop` group can start, stop, and check the status of the SeisComP service without `sudo`.

Result: Both `test1` and `test2` users successfully controlled the service, confirming that the Polkit configuration and environment script functioned as intended. (Full terminal output in Appendix A.5.1).

4.3.2 Test Case 2: File Ownership and Access

Objective: Ensure that operational files generated by SeisComP are owned by the service account and accessible to the group.

Result: All generated files were owned by `seiscomp:sysop` with group read/write permissions, meeting the requirements.(Full terminal output in Appendix A.5.2).

4.3.3 Test Case 3: Custom Systemd Service Check

Objective: Check the custom systemd Service is working and working as planned.

Result: Both custom systemd service is running and functioning as expected (Detailed output in appendix A.5.3).

4.3.4 Test Case 4: Ansible Idempotency and Re-deployment

Objective: Verify that re-running the Ansible roles does not produce unintended changes.

Result: The playbook re-run completed without modifying existing configurations, confirming idempotency (Full terminal output in appendix A.5.4).

4.4 Discussion

The testing outcomes demonstrate that the challenges identified in the initial single-user setup were successfully mitigated through the implemented solution. The introduction of custom systemd services and Polkit rules ensured that service control was securely delegated without requiring root privileges. File permission issues, which had previously restricted multi-user operation, were resolved by enforcing group ownership and access policies through a dedicated custom systemd service that automatically corrects permissions for files generated during application runtime. Furthermore, the Ansible-based automation confirmed that the environment could be reliably reproduced, with idempotency guaranteeing consistency across multiple deployments. Overall, the tests validate that the proposed approach not only addressed the original limitations but also established a secure, maintainable, and scalable framework for multi-user SeisComP management.

In addition to the validation tests described above, a set of real-time operational plots generated by SeisComP (using tools such as `scrttv` and `scheli`) are included in Appendix B). These plots, while not central to the primary objectives of this project, provide a visual demonstration of the system's capability to process and display seismic data in real time, thereby highlighting the practical context of the deployed environment.

4.5 Summary

This chapter evaluated the proposed solution through structured testing. All objectives were achieved, confirming the reliability and security of the multi-user SeisComP environment.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This research project successfully addressed the limitations of the default single-user SeisComP setup by designing and implementing a secure, maintainable multi-user environment. Through a structured methodology that began with manual installation and problem identification, followed by a manually configured group-based solution, and ultimately culminating in a fully automated Ansible deployment, the project achieved its primary objectives.
Key accomplishments include:

- Identification and documentation of operational restrictions in the default SeisComP setup
- Implementation of a shared environment using Linux user groups, file permissions, and ACLs
- Creation of custom `systemd` service units and Polkit rules to delegate service control
- Configuration of environment variables and execution paths for group users
- Development of robust and reusable Ansible roles for automated deployment

The resulting solution enables authorized users to manage SeisComP services securely, without compromising file access or requiring root privileges. The use of automation tools further ensures scalability and reproducibility across environments.

5.2 Future Work

While the current solution meets the core goals of the project, there are several opportunities for further development:

- Integration with centralized authentication systems (e.g., LDAP or Active Directory) to manage user access dynamically
- Enhanced monitoring and alerting mechanisms for service failures and log anomalies
- To manage the size of log files generated by the fix_seiscomp_permission process, a log rotation script can be implemented if required.
- A web-based interface or dashboard for service management and log visualization.
- Integration with containerization tools like Docker to support portable deployment
- Security audits or hardening for environments operating in sensitive domains

These improvements would further increase the usability, flexibility, and maintainability of SeisComP in institutional or production-scale deployments.

5.3 Final Remarks

The project demonstrates the importance of practical system-level design in overcoming real-world software limitations. By combining open-source tools, system administration principles, and configuration automation, a sustainable solution was achieved that empowers collaborative usage of critical infrastructure like SeisComP.

Appendix A

SeisComP Multi-user How TOs

A.1 Manual SeisComP Installation Steps

1. Log in to the Server with a privileged user account and Password.
2. Create a dirctory with the command "`mkdir /opt/download/`", and download the updated SeisComP packages from the mirror.
   ```
   #wget https://www.seiscomp.de/downloader/seiscomp-6.7.6-debian12-i686.tar.gz
   #wget https://www.seiscomp.de/downloader/seiscomp-6.7.6-doc.tar.gz
   #wget https://www.seiscomp.de/downloader/seiscomp-maps.tar.gz
   ```
3. Decompress the packages in the directory "`/opt/`", this will create a folder named "`seiscomp`" in this directory.
   ```
   #tar -zxvf /opt/download/seiscomp-6.7.6-debian12-i686.tar.gz
   #tar -zxvf /opt/download/seiscomp-6.7.6-doc.tar.gz
   #tar -zxvf /opt/download/seiscomp-maps.tar.gz
   ```
4. Go to the directory "`/opt/seiscomp/bin/`" as non root user, and Install dependencies using the "`seiscomp`" command.
   ```
    #seiscomp install-deps base
    #seiscomp install-deps mariadb-server
    #seiscomp install-deps gui fdsnws
   ```
5. Copy the SeisComP environment variables to the directory:**/etc/profile.d/**, and source this file so that normal users can use the SeisComP commands.
   ```
   #/opt/seiscomp/bin/seiscomp print env > /etc/profile.d/seiscomp.sh
   $ source /etc/profile.d/seiscomp.sh
   ```
6. Now secure the mariadb-server, set root password, and carefully select options according to the official documentation[39]
   ```
   # mariadb-secure-installation
   ```
   - `Change the root password? [Y/n] y`
   - `Remove anonymous users? [Y/n] y`
   - `Disallow root login remotely? [Y/n] y`
   - `Remove test database and access to it? [Y/n] y`
   - `Reload privilege tables now? [Y/n] y`
7. It is not advised to run seiscomp as root user, so it is needed to change the ownership of this seiscomp directory "`/opt/seiscomp`"
   ```
     #chown -R subrata:subrata /opt/seiscomp
   ```
8. As a normal user "subrata", run the command "`seiscomp setup`" and select options as shown in the Figure: A.1.

```
subrata@dserver1:/opt/seiscomp$ seiscomp setup

====================================================================
SeisComP setup
====================================================================

This initializes the configuration of your installation.
If you already made adjustments to the configuration files
be warned that this setup will overwrite existing parameters
with default values. This is not a configurator for all
options of your setup but helps to setup initial standard values.

--------------------------------------------------------------------
Hint: Entered values starting with a dot (.) are handled
      as commands. Available commands are:

      quit: Quit setup without modification to your configuration.
      back: Go back to the previous parameter.
      help: Show help about the current parameter (if available).

      If you need to enter a value with a leading dot, escape it
      with backslash, e.g. "\.value".
--------------------------------------------------------------------

Agency ID []: DLR
Datacenter ID []:
Organization string []:
Enable database storage. [yes]:
 0) mysql/mariadb
      MySQL/MariaDB server.
 1) postgresql
      PostgresSQL server version 9 or later.
 2) sqlite3
      SQLite3 database.
Database backend [0]:
Create database [yes]:
MYSQL root password. (input not echoed) []:
Run as super user. [no]:
Drop existing database. [no]:
 0) utf8mb4
 1) utf8
Character set [0]:
Database name. [seiscomp]:
Database hostname. [localhost]:
Database read-write user. [sysop]:
Database read-write password. [sysop]:
Database public hostname. [localhost]:
Database read-only user. [sysop]:
Database read-only password. [sysop]:

Finished setup
--------------

P) Proceed to apply configuration
D) Dump entered parameters
B) Back to last parameter
Q) Quit without changes
Command? [P]: P
```

FIGURE A.1: SeisComP Setup

9. Copy the default configuration files related to `scmaster`, `seedlink`, and `slarchieve` to the directory "`/opt/seiscomp/etc/`". Also enable the modules.
   `$ seiscomp enable scmaster`
   `$ seiscomp enable slarchieve`
   `$ seiscomp enable seedlink`
10. At this point, SeisComP is installed, and it can be tested that the service is running as shown in the figure A.2 as user "subrata" with the commnad `seiscomp status enabled`.

```
subrata@dserver1:/opt$ seiscomp status enabled
scmaster             is running
seedlink             is running
slarchive            is running
Summary: 3 modules enabled
subrata@dserver1:/opt$
```

FIGURE A.2: SeisComP Status

A.2 Dissection of the problem

The initial installation was performed by the user account `subrata`, resulting in all files within the installation directory being owned by this user, as reflected in the file permission settings A.3.

```
subrata@dserver1:/opt$ ls -lh
total 8.0K
drwxr-xr-x 3 subrata subrata 4.0K Jun  2 09:30 download
drwxr-xr-x 9 subrata subrata 4.0K Jun  2 09:24 seiscomp
subrata@dserver1:/opt$ ls -lh seiscomp/
total 28K
drwxr-xr-x  2 subrata subrata 4.0K May 15 22:37 bin
drwxr-xr-x  7 subrata subrata 4.0K Jun  2 09:47 etc
drwxr-xr-x  6 subrata subrata 4.0K May 15 22:37 include
drwxr-xr-x  4 subrata subrata 4.0K May 15 22:37 lib
drwxr-xr-x  2 subrata subrata 4.0K May 15 22:37 sbin
drwxr-xr-x 20 subrata subrata 4.0K Nov 22  2012 share
drwxr-xr-x  5 subrata subrata 4.0K Jun  2 09:49 var
subrata@dserver1:/opt$
```

FIGURE A.3: SeisComP directory permission

To identify the problem, two test users—`user1` and `user2`—were created and added to the `sysop` group. The file permissions originally assigned to the primary user `subrata` were then replicated for the group, ensuring that all members, including the test users, had identical access rights A.4.

```
root@dserver1:/opt# ll
total 8.0K
drwxr-xr-x 3 subrata subrata 4.0K Jun  2 09:30 download
drwxr-xr-x 9 subrata subrata 4.0K Jun  2 09:24 seiscomp
root@dserver1:/opt# chown -R subrata:sysop seiscomp
root@dserver1:/opt# chmod -R g=u seiscomp
root@dserver1:/opt# ll
total 8.0K
drwxr-xr-x 3 subrata subrata 4.0K Jun  2 09:30 download
drwxrwxr-x 9 subrata sysop   4.0K Jun  2 09:24 seiscomp
root@dserver1:/opt#
```

FIGURE A.4: SeisComP Modified Directory Permission

Despite having the same group and file permissions as the user `subrata`, the test users encountered a warning (A.5) when attempting to run the command `seiscomp status enable`, indicating they were unable to retrieve the service status.

```
user1@dserver1:~$ seiscomp status enabled
scmaster                  is not running [WARNING]
seedlink                  is not running [WARNING]
slarchive                 is not running [WARNING]
Summary: 3 modules enabled
user1@dserver1:~$
```

```
user2@dserver1:~$ seiscomp status enabled
scmaster                  is not running [WARNING]
seedlink                  is not running [WARNING]
slarchive                 is not running [WARNING]
Summary: 3 modules enabled
user2@dserver1:~$
```

FIGURE A.5: SeisComP Status as Test Users

This limitation persisted even though ownership and permissions appeared consistent. Upon examining the directory `/opt/seiscomp/var/run/` (A.6), it was found that SeisComP generates certain runtime files during execution, which are owned by the user who initiates the service—thereby restricting access to others.

```
root@dserver1:/opt/seiscomp/var/run# ll
total 20K
-rw-r--r-- 1 subrata subrata    4 Jun  2 12:55 scmaster.pid
-rw-r--r-- 1 subrata subrata    0 Jun  2 12:55 scmaster.run
-rw-r--r-- 1 subrata subrata    5 Jun  2 12:55 seedlink.pid
-rw-r--r-- 1 subrata subrata    0 Jun  2 12:55 seedlink.run
-rw-rw-r-- 1 subrata sysop      5 Jun  2 12:55 seiscomp.pid
drwxr-xr-x 2 subrata subrata 4.0K Jun  2 12:55 slarchive
-rw-r--r-- 1 subrata subrata    5 Jun  2 12:55 slarchive.pid
-rw-r--r-- 1 subrata subrata    0 Jun  2 12:55 slarchive.run
root@dserver1:/opt/seiscomp/var/run#
```

FIGURE A.6: /seiscomp/var/run directory

After stopping the service and modifying the directory permissions to match the group settings, the service was started using the user `user1` A.7. It was observed that the service ran successfully under this user .

```
user1@dserver1:~$ seiscomp start
starting scmaster
starting seedlink
 maximum number of open files set to 1048576
starting slarchive
Summary: 3 modules started
user1@dserver1:~$ seiscomp status enabled
scmaster                 is running
seedlink                 is running
slarchive                is running
Summary: 3 modules enabled
user1@dserver1:~$
```

```
user2@dserver1:~$ seiscomp status enabled
scmaster                 is not running [WARNING]
seedlink                 is not running [WARNING]
slarchive                is not running [WARNING]
Summary: 3 modules enabled
user2@dserver1:~$
```

```
subrata@dserver1:~$ seiscomp status enabled
scmaster                 is not running [WARNING]
seedlink                 is not running [WARNING]
slarchive                is not running [WARNING]
Summary: 3 modules enabled
subrata@dserver1:~$
```

FIGURE A.7: Running as user1

However, the ownership of the generated runtime files was now associated with `user1`, resulting in access failures when the status was checked by `user2` or `subrata`. A further inspection of the `/seiscomp/var/run/` directory confirmed that the newly created files were owned by the user who initiated the service A.8.

```
root@dserver1:/opt/seiscomp/var/run# ll
total 20K
-rw-r--r-- 1 user1   user1      4 Jun  2 13:08 scmaster.pid
-rw-r--r-- 1 user1   user1      0 Jun  2 13:08 scmaster.run
-rw-r--r-- 1 user1   user1      5 Jun  2 13:08 seedlink.pid
-rw-r--r-- 1 user1   user1      0 Jun  2 13:08 seedlink.run
-rw-rw-r-- 1 subrata sysop      5 Jun  2 13:12 seiscomp.pid
drwxr-xr-x 2 subrata subrata 4.0K Jun  2 13:01 slarchive
-rw-r--r-- 1 user1   user1      5 Jun  2 13:08 slarchive.pid
-rw-r--r-- 1 user1   user1      0 Jun  2 13:08 slarchive.run
root@dserver1:/opt/seiscomp/var/run#
```

FIGURE A.8: /seiscomp/var/run directory as user1

A.3 Ansible Automation Multi-user SeisComP Solution

A.3.1 Ansible Control Node Preparation

This section contains the full “how-to” instructions for preparing the Ansible control node, configuring SSH authentication, and encrypting sensitive data using Ansible Vault.

A.3.2 Installing Ansible

```
sudo apt update
sudo apt install ansible -y
```

A.3.3 Project Workspace

A directory `/home/myproject` was created containing:

- `ansible.cfg` (Figure A.9)

```
[defaults]
inventory = ./inventory.yml
roles_path = ./roles
```

FIGURE A.9: Code Snippet - ansible.cfg

- `inventory.yml` (Figure A.10)

```
all:
  children:
    dev_servers:
      hosts:
        dserver1.0:
          ansible_host: 192.168.101.201
        dserver2.0:
          ansible_host: 192.168.101.202
```

FIGURE A.10: Code Snippet - inventory.yml

A.3.4 SSH Key Authentication

```
ssh-keygen -t ed25519
ssh-copy-id -i ~/.ssh/id_ed25519.pub subrata@192.168.101.201
ssh-copy-id -i ~/.ssh/id_ed25519.pub subrata@192.168.101.202
```

A.3.5 Ansible Vault Encryption

Before encryption, the file `group_vars/all/vault.yaml` contained sensitive data in plaintext (Figure A.11).

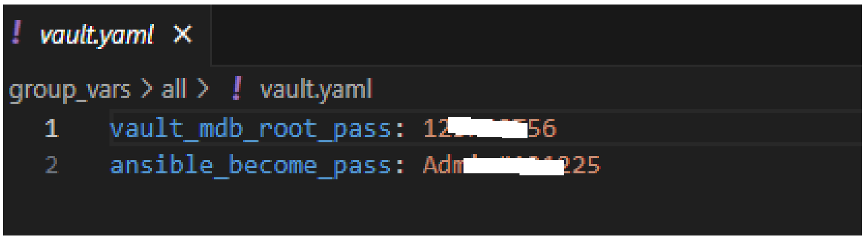


FIGURE A.11: Ansible vault in plaintext

Encryption was applied with:
`ansible-vault encrypt group_vars/all/vault.yaml`
After encryption, the contents became unreadable without the vault password (Figure A.12).

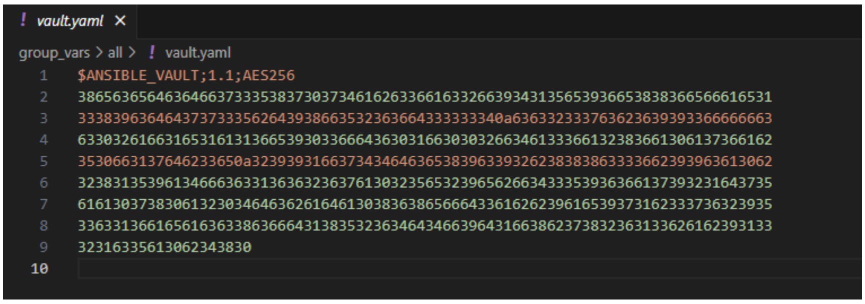


FIGURE A.12: Ansible vault encrypted

A.3.6 MariaDB Role

The `mariadb` role follows standard Ansible best practices to maintain modularity, clarity, and security in its implementation.

Directory Structure and Code: The role's structure is organized into purpose-specific components:

- `defaults/main.yml` – Defines default variables, including package names, service identifiers, and configuration parameters for MariaDB installation (Figure A.13).

```
mariadb_packages:
  - python3-pymysql
  - mariadb-server
  - mariadb-client
mariadb_service: mariadb
```

FIGURE A.13: MariaDB Role – defaults/main.yml

- `handlers/main.yml` – Contains handlers to manage the MariaDB service, such as restarting or reloading after configuration changes(Figure A.14).

```
- name: Restart MariaDB Service
  ansible.builtin.systemd:
    name: "{{ mariadb_service }}"
    state: restarted
```

FIGURE A.14: MariaDB Role – handlers/main.yml

- `tasks/main.yml` – Implements the main installation workflow, covering package installation, service configuration, and initial security hardening (Figure A.15).

```
        - name: Update APT package index
          ansible.builtin.apt:
            update_cache: true
        - name: Install MariaDB and dependencies
          ansible.builtin.apt:
            name: "{{ mariadb_packages }}"
            state: present
        - name: Enable and start MariaDB service
          ansible.builtin.systemd:
            name: "{{ mariadb_service }}"
            enabled: true
            state: started
        - name: Check if root can connect via socket
          community.mysql.mysql_info:
            login_unix_socket: /var/run/mysqld/mysqld.sock
          register: mysql_info_socket
          ignore_errors: true
          become: true
        - name: Secure MariaDB Installation (Set root password)
          community.mysql.mysql_user:
            name: root
            password: "{{ vault_mdb_root_pass }}"
            login_unix_socket: /var/run/mysqld/mysqld.sock
          become: true
          when: mysql_info_socket is succeeded
        - name: Remove anonymous users
          community.mysql.mysql_user:
            name: ''
            host_all: yes
            state: absent
            login_user: root
            login_password: "{{ vault_mdb_root_pass }}"
        - name: Removing test database
          community.mysql.mysql_db:
            name: test
            state: absent
            login_user: root
            login_password: "{{ vault_mdb_root_pass }}"
        - name: Disable remote root login
          community.mysql.mysql_user:
            name: root
            host: "{{ item }}"
            state: absent
            login_user: root
            login_password: "{{ vault_mdb_root_pass }}"
          loop:
            - "%"
            - "::1"

```

FIGURE A.15: Code Snippets: mariadb/tasks/main.yml

- `group_vars/all/vault.yml` – Stores sensitive information as described in (Figure A.12)

This separation of concerns allows for clear role functionality, facilitates reuse in other projects, and ensures that critical credentials remain securely stored.

A.3.7 SeisComP Role

The SeisComP role is designed following established Ansible best practices to ensure modularity, maintainability, and operational security during deployment.

Directory Structure and Code Snippets: The role's organization is segmented into function-specific components, enabling clear separation of concerns:

- **defaults/main.yml** – Defines essential variables for the SeisComP installation, including package lists, source URLs, and configuration parameters required to set up the environment.

```
# Seiscomp packages and defining variables
---
# SeisComP repository URL
seiscomp_version: "6.7.9"
seiscomp_repo_url: "https://www.seiscomp.de/downloader/seiscomp-{{
↪ seiscomp_version }}-debian12-x86_64.tar.gz"
#seiscomp_repo_url_full: "https://www.seiscomp.de/downloader/seiscomp
↪ -6.7.9-debian12-x86_64.tar.gz"

# SeisComP working and installed directories
seiscomp_working_dir: "/opt/download"
seiscomp_installed_dir: "/opt"

# SeisComP user and group
seiscomp_user: "seiscomp"
seiscomp_group: "sysop"

# Seiscomp Dependencies and base packages
seiscomp_base_packages_with_gui:
  - at-spi2-core
  - gcc
  - gfortran
  - inotify-tools
  - libblas3
  - libboost-filesystem1.74.0
  - libboost-iostreams1.74.0
  - libboost-program-options1.74.0
  - libboost-regex1.74.0
  - libboost-system1.74.0
  - libboost-thread1.74.0
  - libgfortran5
  - liblapack3
  - libmariadb3
  - libncurses5
  - libpq5
  - libpython3-dev
  - libpython3.11
  - libqt5gui5
  - libqt5opengl5
  - libqt5printsupport5
  - libqt5sql5
  - libqt5sql5-sqlite
  - libqt5svg5
  - libqt5xml5
  - libquadmath0
  - libssl3
  - libtinfo5
  - libxml2
  - mariadb-common
  - mysql-common
  - pkexec
  - python3-dev
  - python3-numpy
  - python3-pexpect
  - python3-pip
  - python3-pytest


```

FIGURE A.16: Code Snippets: seiscomp/defaults/main.yml

- **files/** – Contains static files that must be copied to specific target directories as depicted figure A.17 to ensure correct SeisComP configuration and operation.

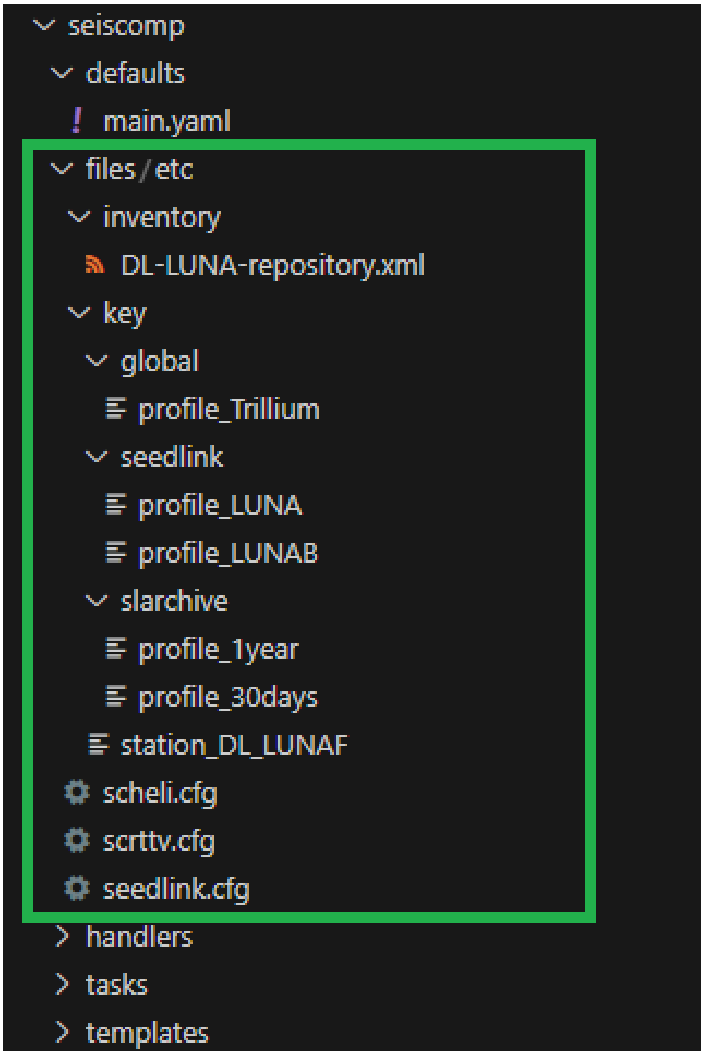


FIGURE A.17: Code Snippets: seiscomp files directory

- **handlers/main.yml** – Provides handlers to manage SeisComP services, allowing controlled restarts or reloads when configuration changes occur (Figure A.18).

```
# Handlers for SeisComP role
- name: Reload and (re)start fix_seiscomp_permission service
  ansible.builtin.systemd:
    name: fix_seiscomp_permission.service
    #daemon_reload: true
    enabled: true
    state: started

# restart polkit service
- name: Enable and Restart polkit service
  ansible.builtin.systemd:
    name: polkit.service
    enabled: yes
    state: restarted

# Enable and restart SeisComP service
- name: Enable and restart SeisComP service
  ansible.builtin.systemd:
    name: seiscomp.service
    enabled: yes
    state: restarted

# Daemon reload once changes are made to the service files
- name: Daemon_Reload
  become: true
  ansible.builtin.systemd:
    name: seiscomp.service
    daemon_reload: true
```

FIGURE A.18: Code Snippets: seiscomp/handlers/main.yml

- **tasks/main.yml** – Implements the complete workflow for installing SeisComP, covering package retrieval, installation, configuration, and integration steps.

```
---
# 0) Ensure MariaDB is present and accessible (or install it if not present)
- name: Check if MariaDB package is installed
  ansible.builtin.shell: dpkg -l | grep -qw mariadb-server
  register: mariadb_installed
  failed_when: false
  changed_when: false
  become: true

- name: Check MariaDB root access
  community.mysql.mysql_info:
    login_user: root
    login_password: "{{ vault_mdb_root_pass }}"
  register: mariadb_accessible
  failed_when: false
  changed_when: false
  when: mariadb_installed.rc == 0
  become: true

- name: Install & configure MariaDB (on-demand)
  ansible.builtin.include_role:
    name: mariadb
  when: mariadb_installed.rc != 0 or (mariadb_accessible is defined and
    ↪ mariadb_accessible.failed)

# Debug message
- name: This playbook is executed by...
  debug:
    msg: " The playbook is executed by {{ ansible_user_id }}"

# 1) User / Group / Directories
- name: Create a system user for SeisComP
  ansible.builtin.user:
    name: "{{ seiscomp_user }}"
    comment: "Seiscomp System User"
```

```
    shell: /bin/bash
    create_home: yes
    home: "/home/{{ seiscomp_user }}"
    state: present

- name: Ensure group {{ seiscomp_group }} exists
  ansible.builtin.group:
    name: "{{ seiscomp_group }}"
    state: present

- name: Make sure working and installed directories exist
  ansible.builtin.file:
    path: "{{ item }}"
    state: directory
    owner: "{{ seiscomp_user }}"
    group: "{{ seiscomp_group }}"
    mode: "ug=rwx,o=r"
  loop:
    - "{{ seiscomp_working_dir }}"
    - "{{ seiscomp_installed_dir }}"

# 2) Download & extract SeisComP archive (if needed)
- name: Check if SeisComP archive already exists
  ansible.builtin.stat:
    path: "{{ seiscomp_working_dir }}/seiscomp-{{ seiscomp_version }}-debian12-x86_64.tar.gz"
  register: seiscomp_archive
  changed_when: false

- name: Download SeisComP archive
  ansible.builtin.get_url:
    url: "{{ seiscomp_repo_url }}"
    dest: "{{ seiscomp_working_dir }}/seiscomp-{{ seiscomp_version }}-debian12-x86_64.tar.gz"
    mode: "ug=rwx,o=r"
    owner: "{{ seiscomp_user }}"
    group: "{{ seiscomp_group }}"
    force: false
  when: not seiscomp_archive.stat.exists
  register: seiscomp_download

- name: Extract Seiscomp
  ansible.builtin.unarchive:
    src: "{{ seiscomp_working_dir }}/seiscomp-{{ seiscomp_version }}-debian12-x86_64.tar.gz"
    dest: "{{ seiscomp_installed_dir }}"
    remote_src: true
    owner: "{{ seiscomp_user }}"
    group: "{{ seiscomp_group }}"
    mode: "ug=rwx,o=r"
    extra_opts: [--same-owner]
    creates: "{{ seiscomp_installed_dir }}/seiscomp/bin/seiscomp"

- name: Install dependencies base with gui
  ansible.builtin.apt:
    name: "{{ seiscomp_base_packages_with_gui }}"
    state: present
    update_cache: true
  become: true

# 3) Permissions / ACLs
- name: Set ACLs for seiscomp directory
  ansible.posix.acl:
    path: "{{ seiscomp_installed_dir }}/seiscomp"
    entity: "{{ seiscomp_user }}"
    etype: user
    permissions: rwx
    default: true
    state: present
    recursive: true

- name: Set default ACLs for user
```

```
  ansible.posix.acl:
    path: "{{ seiscomp_installed_dir }}/seiscomp"
    entity: "{{ seiscomp_user }}"
    etype: user
    permissions: rwx
    default: true
    state: present
    recursive: true

- name: Set default ACLs for group
  ansible.posix.acl:
    path: "{{ seiscomp_installed_dir }}/seiscomp"
    entity: "{{ seiscomp_group }}"
    etype: group
    permissions: rwx
    default: true
    state: present
    recursive: true

# 4) Copy templates / service unit files / polkit / env wrappers
- name: Copy files for services and for {{ seiscomp_group }} group
  ansible.builtin.template:
    src: "{{ item }}.j2"
    dest: "/{{ item }}"
    owner: root
    group: root
    mode: "0644"
  loop:
    - etc/systemd/system/seiscomp.service
    - etc/systemd/system/fix_seiscomp_permission.service
    - etc/polkit-1/rules.d/50-seiscomp.rules
    - etc/profile.d/seiscomp.sh
    - etc/profile.d/luna-seiscomp.sh
  notify: Daemon_Reload

- name: Copy fix_seiscomp_permission bash script
  ansible.builtin.template:
    src: usr/local/bin/fix_seiscomp_permission.sh.j2
    dest: /usr/local/bin/fix_seiscomp_permission.sh
    mode: '0755'
    owner: root
    group: root

- name: Check if log file for fix_seiscomp_permission exists
  ansible.builtin.stat:
    path: /var/log/fix_seiscomp_permission.log
  register: fix_seiscomp_permission

- name: Create log file, and has correct ownership & permissions
  ansible.builtin.file:
    path: /var/log/fix_seiscomp_permission.log
    state: touch
    owner: root
    group: "{{ seiscomp_group }}"
    mode: 'u=rw,g=rw,o=r'
    modification_time: preserve
  when: not fix_seiscomp_permission.stat.exists
  notify: Reload and (re)start fix_seiscomp_permission service

- name: Ensure SeisComP wrapper is sourced in all interactive shells to work
    ↪ start/stop/restart
  ansible.builtin.blockinfile:
    path: /etc/bash.bashrc
    create: true
    insertafter: EOF
    marker: "# {mark} ANSIBLE MANAGED: myseiscomp wrapper"
    block: |-
      # Source SeisComP wrapper for sysop users
      if [ -f /etc/profile.d/luna-seiscomp.sh ]; then
        source /etc/profile.d/luna-seiscomp.sh
      fi
    mode: '0644'
```

```yaml
    owner: root
    group: root

# 5) SeisComP interactive setup (guarded with a marker file so it runs only once )
- name: Check if SeisComP setup already completed
  ansible.builtin.stat:
    path: "{{ seiscomp_installed_dir }}/seiscomp/.setup_done"
  register: seiscomp_setup_marker
  changed_when: false
  become: true

- name: Run SeisComP interactive setup
  ansible.builtin.expect:
    command: "{{ seiscomp_installed_dir }}/seiscomp/bin/seiscomp setup "
    responses:
      "Agency ID.* ": "DLR"
      "Datacenter.* ": ""
      "Organization string.* ": "MUSCDLR"
      "Enable database.* ": "yes"
      "Database backend.* ": "0"
      "Create database.* ": "yes"
      "MYSQL root password.* ": "{{ vault_mdb_root_pass }}"
      "Run as super user.* ": "no"
      "Drop existing database.* ": "yes"
      " .* utf8mb4.* ": "0"
      "Database name.* ": "seiscomp"
      "Database hostname.* ": "localhost"
      "Database read-write.* ": "sysop"
      "Database read-write password.*  ": "sysop"
      "Database public hostname.* ": "localhost"
      "Database read-only user.* ": "sysop"
      "Database read-only password.* ": "sysop"
      " .* Proceed to.* ": "P"
  become: true
  become_user: "{{ seiscomp_user }}"
  when: not seiscomp_setup_marker.stat.exists
  changed_when: false

- name: Create SeisComP setup marker file
  ansible.builtin.file:
    path: "{{ seiscomp_installed_dir }}/seiscomp/.setup_done"
    state: touch
    owner: "{{ seiscomp_user }}"
    group: "{{ seiscomp_group }}"
    mode: '0664'
  when: not seiscomp_setup_marker.stat.exists

# 6) Copy application configuration files
- name: Check if configuration files copied already
  ansible.builtin.stat:
      path: "{{ seiscomp_installed_dir}}/seiscomp/etc/inventory/DL-LUNA-repository.xml"
  register: lunaconfig_files

- name: Copy Seiscomp Config files
  ansible.builtin.copy:
    src: "etc/"
    dest: "{{ seiscomp_installed_dir }}/seiscomp/etc"
    owner: "{{ seiscomp_user }}"
    group: "{{ seiscomp_group }}"
    mode: "0664"
    directory_mode: "0775"
    force: true
    remote_src: false
  when: not lunaconfig_files.stat.exists

# 7) Enable modules, update config and ensure service state
- name: Enable update-config and start seiscomp service
  ansible.builtin.shell: |
    {{ seiscomp_installed_dir}}/seiscomp/bin/seiscomp enable scmaster
    {{ seiscomp_installed_dir}}/seiscomp/bin/seiscomp enable seedlink
```

```
      {{ seiscomp_installed_dir}}/seiscomp/bin/seiscomp enable slarchive
      {{ seiscomp_installed_dir}}/seiscomp/bin/seiscomp update-config
      {{ seiscomp_installed_dir}}/seiscomp/bin/seiscomp stop
    become: true
    become_user: seiscomp
    when: not lunaconfig_files.stat.exists
    notify:
      - Enable and Restart polkit service
      - Enable and restart SeisComP service
# End of tasks/main.yml
```

FIGURE A.19: Code Snippets: seiscomp/tasks/main.yml

- **templates/** – Holds Jinja2-based templates used to dynamically generate configuration scripts from predefined variables and place them into appropriate system paths, as can be observed from the directory from figure A.20.

FIGURE A.20: Code Snippets: seiscomp templates directory

The content of `templates/etc/polkit/rules.d/50-seiscomp.rules.j2` file:

```
    polkit.addRule(function(action, subject) {
        if (
            action.id == "org.freedesktop.systemd1.manage-units" &&
            action.lookup("unit") == "seiscomp.service" &&
            subject.isInGroup("{{ seiscomp_group }}")
        ) {
            return polkit.Result.YES;

```

FIGURE A.21: Code Snippets: 50-seiscomp.rules.j2

The content of `templates/etc/profile.d/seiscomp.sh.j2` file:

```
# Generated by Ansible - DO NOT EDIT MANUALLY
export SEISCOMP_ROOT="{{ seiscomp_installed_dir }}/seiscomp"
export PATH="{{ seiscomp_installed_dir }}/seiscomp/bin:$PATH"
export LD_LIBRARY_PATH="{{ seiscomp_installed_dir }}/seiscomp/lib:$LD_LIBRARY_PATH"
export PYTHONPATH="{{ seiscomp_installed_dir }}/seiscomp/lib/python:$PYTHONPATH"
export MANPATH="{{ seiscomp_installed_dir }}/seiscomp/share/man:$MANPATH"
source "{{ seiscomp_installed_dir }}/seiscomp/share/shell-completion/seiscomp.bash"

```

FIGURE A.22: Code Snippets: seiscomp.sh.j2

The content of `templates/etc/profile.d/luna-seiscomp.sh.j2` file:

```
#!/usr/bin/env bash

# Only apply to users in '{{seiscomp_group}}' group
if id -nG "$USER" | grep -qw {{seiscomp_group}}; then

  # Source the main SeisComP environment
  [ -f /etc/profile.d/seiscomp.sh ] && source /etc/profile.d/seiscomp.sh

  # Override seiscomp command using systemctl
  seiscomp() {
    case "$1" in
      restart) systemctl restart seiscomp.service     ;;
      start)   systemctl start seiscomp.service       ;;
      stop)    systemctl stop seiscomp.service        ;;
      *)       /opt/seiscomp/bin/seiscomp "$@"        ;;
    esac
  }

  export -f seiscomp
fi

```

FIGURE A.23: Code Snippets: luna-seiscomp.sh.j2

- **group_vars/all/vault.yml** – Stores sensitive information as described in (Figure A.12).

A.4 Ansible Playbook

The `pb_install_service.yml` playbook (Figure A.24) was used to deploy the MariaDB and SeisComP roles.

```
---
- name: Install and configure services
  hosts: all
  become: true
  gather_facts: yes
  roles:
    - role: mariadb
      tags: mariadb
    - role: seiscomp
      tags: seiscomp
```

FIGURE A.24: Ansible playbook to install and configure services

A.5 Test Outputs

A.5.1 Test Case 1: Service Control by Group Members and Polkit Authorization Validation

Figure A.25 illustrates the group memberships of the test users `test1` and `test2`, as well as their ability to start, stop, restart, and check the status of the SeisComP service. These results confirm that both users, as members of the `sysop` group, can manage the service without root privileges, while Polkit operates in the background to enforce authorization securely.

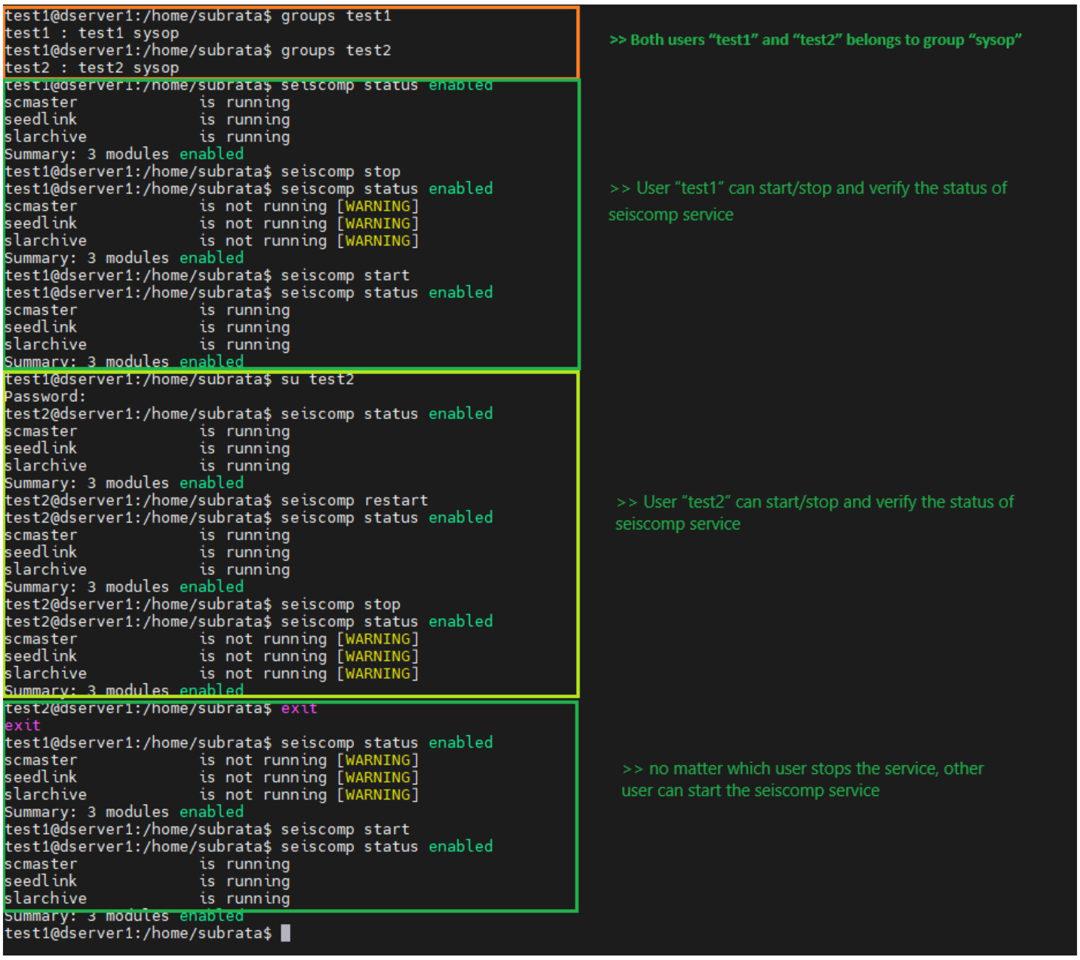


FIGURE A.25: Multi-user Test Cases

A.5.2 Test Case 2: File Ownership and Access

Figure A.26 provides an example from the directory `/opt/seiscomp/var/run/`, showing that files generated during service operation are consistently owned by the service account `seiscomp` and assigned to the group `sysop`. While only one directory is displayed, it reflects the broader system behavior: regardless of which authorized user initiates the service, file ownership and permissions remain aligned with the intended group-based access model, ensuring secure and collaborative multi-user operation.

```
test1@dserver1:/opt/seiscomp/var/run$ ls -lah
total 48K
drwxrwxr--+ 4 seiscomp sysop 4.0K Jul 14 14:59 .
drwxrwxr--+ 5 seiscomp sysop 4.0K Jul  3 17:14 ..
-rw-rw-r--+ 1 seiscomp sysop    5 Aug 11 13:34 scmaster.pid
-rw-rw-r--+ 1 seiscomp sysop    0 Aug 11 13:34 scmaster.run
drwxrwxr--+ 2 seiscomp sysop 4.0K Jul  3 17:15 seedlink
-rw-rw-r--+ 1 seiscomp sysop    6 Aug 11 13:34 seedlink.pid
-rw-rw-r--+ 1 seiscomp sysop    0 Aug 11 13:34 seedlink.run
-rw-rw-r--+ 1 seiscomp sysop    0 Jul  3 17:12 seiscomp.init
-rw-rw-r--+ 1 seiscomp sysop    6 Aug 11 13:34 seiscomp.pid
drwxrwxr--+ 2 seiscomp sysop 4.0K Jul  3 17:15 slarchive
-rw-rw-r--+ 1 seiscomp sysop    6 Aug 11 13:34 slarchive.pid
-rw-rw-r--+ 1 seiscomp sysop    0 Aug 11 13:34 slarchive.run
test1@dserver1:/opt/seiscomp/var/run$
```

FIGURE A.26: System Generated File Ownership

A.5.3 Test Case 3: Custom Systemd Service check

Figure A.27 shows that Custom systemd `fix_seiscomp_permission.service` is active,

```
test1@dserver1:/home/subrata$ systemctl status fix_seiscomp_permission.service
● fix_seiscomp_permission.service - Fix Permission Service
     Loaded: loaded (/etc/systemd/system/fix_seiscomp_permission.service; enabled; preset: enabled)
     Active: active (running) since Mon 2025-08-11 22:11:50 CEST; 5min ago
   Main PID: 216311 (fix_seiscomp_pe)
      Tasks: 16 (limit: 4590)
     Memory: 4.0M
        CPU: 327ms
     CGroup: /system.slice/fix_seiscomp_permission.service
             ├─216311 /bin/bash /usr/local/bin/fix_seiscomp_permission.sh
             ├─216312 /bin/bash /usr/local/bin/fix_seiscomp_permission.sh
             ├─216313 inotifywait -m -r -e create,modify,move /opt/seiscomp/ --format %w%f
             ├─216314 /bin/bash /usr/local/bin/fix_seiscomp_permission.sh
             ├─216467 /bin/bash /usr/local/bin/fix_seiscomp_permission.sh
             ├─216470 /bin/bash /usr/local/bin/fix_seiscomp_permission.sh
             ├─216476 inotifywait -m -r -e create,modify,move /opt/seiscomp/var/lib/archive --format %w%f
             ├─216477 /bin/bash /usr/local/bin/fix_seiscomp_permission.sh
             ├─216478 inotifywait -m -r -e create,modify,move /opt/seiscomp/var/run/slarchive --format %w%f
             ├─216479 /bin/bash /usr/local/bin/fix_seiscomp_permission.sh
             ├─216481 /bin/bash /usr/local/bin/fix_seiscomp_permission.sh
             ├─216484 inotifywait -m -r -e create,modify,move /opt/seiscomp/var/lib/seedlink/buffer --format %w%f
             ├─216485 /bin/bash /usr/local/bin/fix_seiscomp_permission.sh
             ├─216491 /bin/bash /usr/local/bin/fix_seiscomp_permission.sh
             ├─216493 inotifywait -m -r -e create,modify,move /opt/seiscomp/var/lib/seedlink/buffer/DL.LUNAF --format %w%f
             └─216494 /bin/bash /usr/local/bin/fix_seiscomp_permission.sh
test1@dserver1:/home/subrata$
```

FIGURE A.27: systemctl fix_permission_service status

while Figure A.28 displays its log file, confirming that permission adjustments are applied whenever SeisComP generates new files.

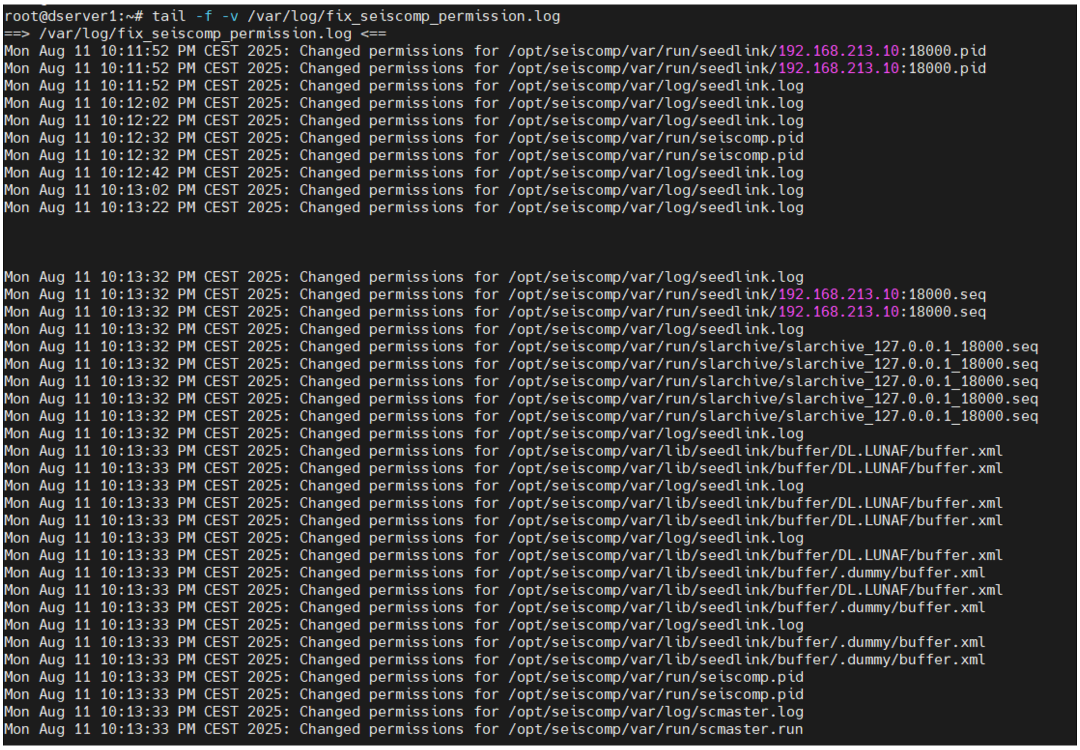


FIGURE A.28: fix_permission_service log generation

Finally, the Figure A.29 shows the status of the `seiscomp.service`, indicating that the application itself is running under the designated system user. Together, these results demonstrate that the custom services not only ensure SeisComP starts reliably but also maintain consistent file ownership and permissions, thereby resolving the operational challenges identified in the manual setup.

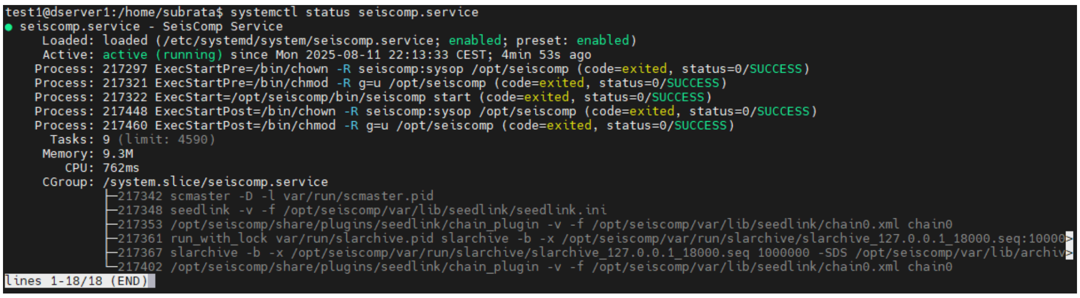


FIGURE A.29: systemctl seiscomp status

A.5.4 Test Case 4: Ansible Idempotency and Re-deployment

Figure A.30 to Figure A.32 demonstrate the idempotent behaviour of the `mariadb` Ansible role. During the first execution, the role successfully installed MariaDB, applied the secure configuration steps, and completed all associated tasks.
When the same playbook was executed immediately afterwards, Ansible detected that no configuration changes were required and therefore skipped any modifications, confirming that the role is idempotent. This behaviour is crucial in automated system administration, as it ensures that repeated executions maintain the desired system state without introducing unnecessary changes or downtime.

FIGURE A.30: First execution of the `mariadb` role showing all installation and configuration steps.

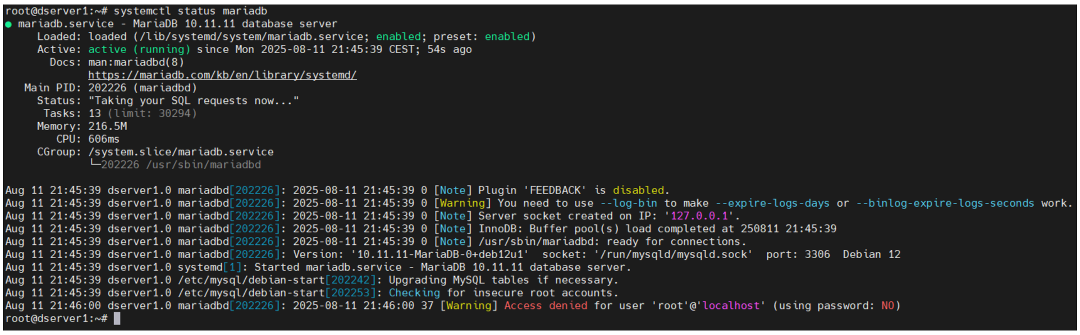


FIGURE A.31: Systemctl Mariadb Status

FIGURE A.32: Second execution of the `mariadb` role showing idempotency.

In the same way, during the first execution of the seiscomp role, it was run successfully to install SeisComP Figure A.33, apply the necessary configurations, and complete all associated tasks to work its multi-user setup.

Again, when the same playbook was executed immediately, Ansible detected that no configuration changes were required and therefore skipped any modifications Figure A.34, confirming that the role is idempotent.

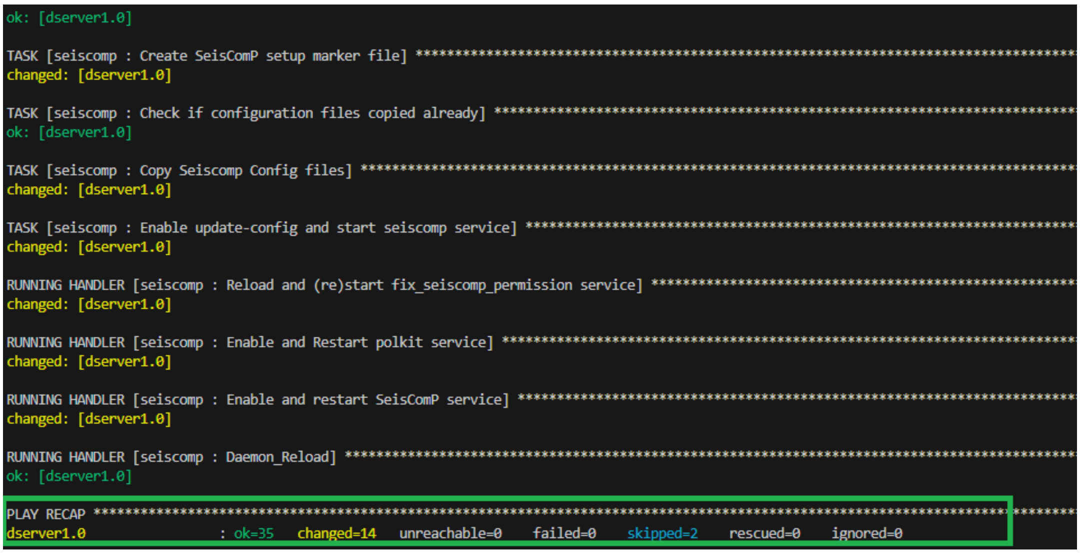


FIGURE A.33: First execution of the `seiscomp` role showing all installation and configuration steps.

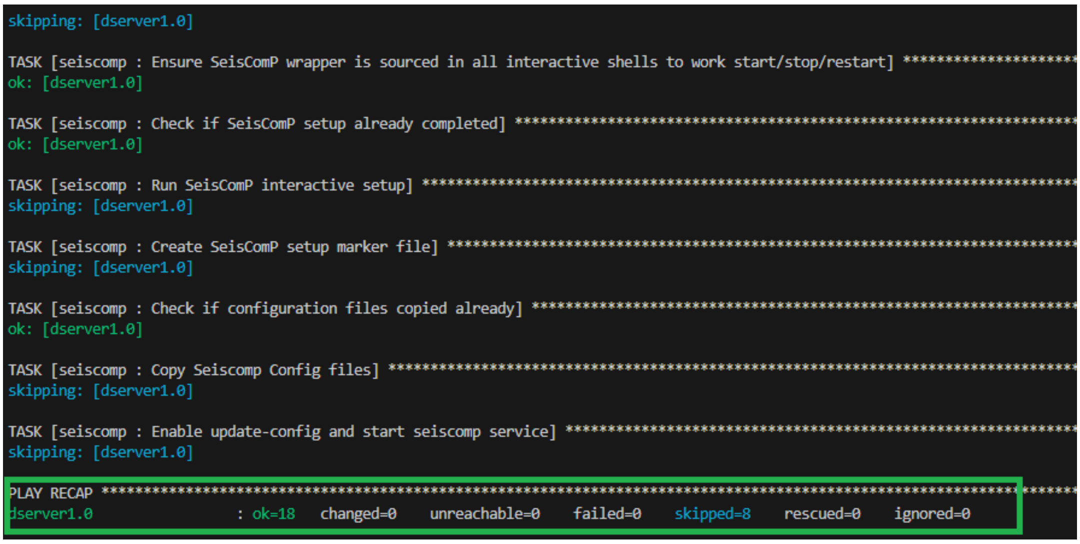


FIGURE A.34: Second execution of the `seiscomp` role showing idemopotency.

Appendix B

Real-Time SeisComP Data Plots

This appendix presents some real-time outputs from SeisComP visualization tools such as `scrttv` and `scheli`. These figures illustrate the system's ability to display real-time seismic data streams. Although these outputs are not part of the core evaluation criteria for the multi-user configuration, they demonstrate that the deployed environment is fully operational and capable of supporting real-world use cases.

B.1 Scheli Plot

The `scheli` tool provides a helicorder-style display of seismic waveform data, where time progresses horizontally and consecutive traces are stacked vertically. The `scheli` plot presented in Figure B.1 illustrates seismic waveforms recorded on the LUNA server during two significant earthquakes that occurred in Zulia, Venezuela on 24th and 25th September, 2025 (magnitude 6.3 and 6.2, see Figure B.2) according to website https://www.emsc-csem.org.

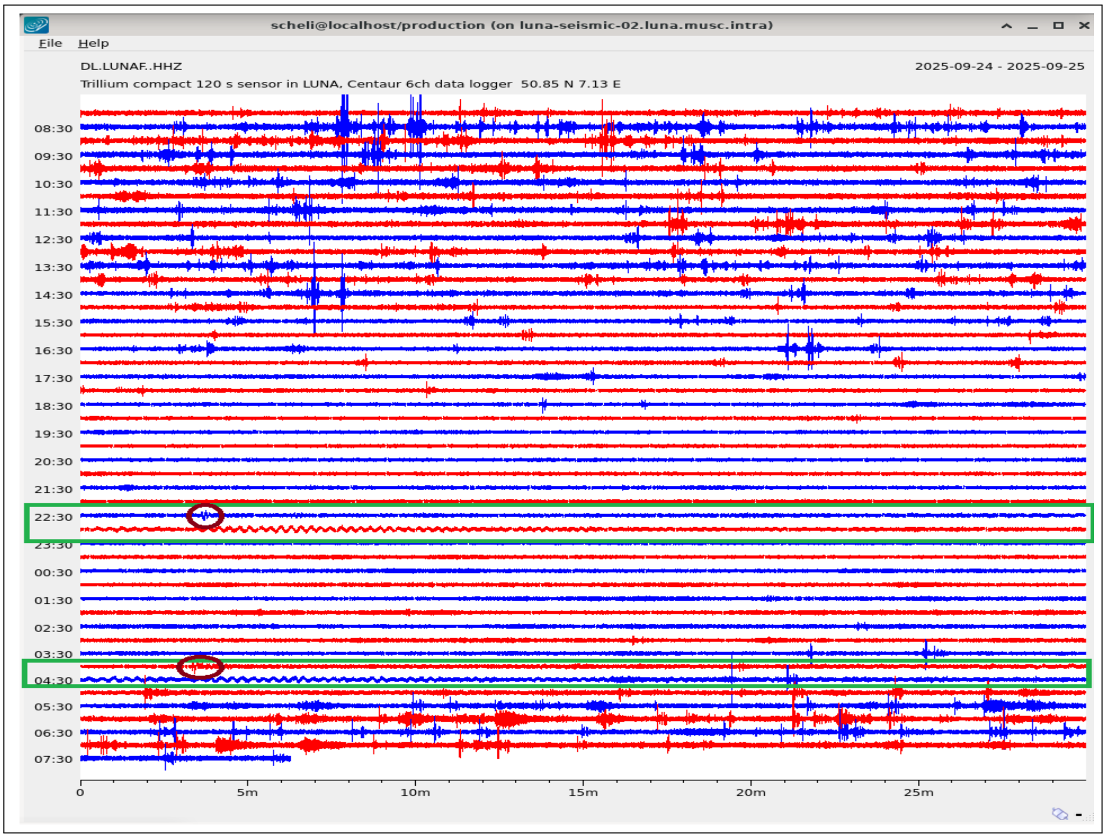
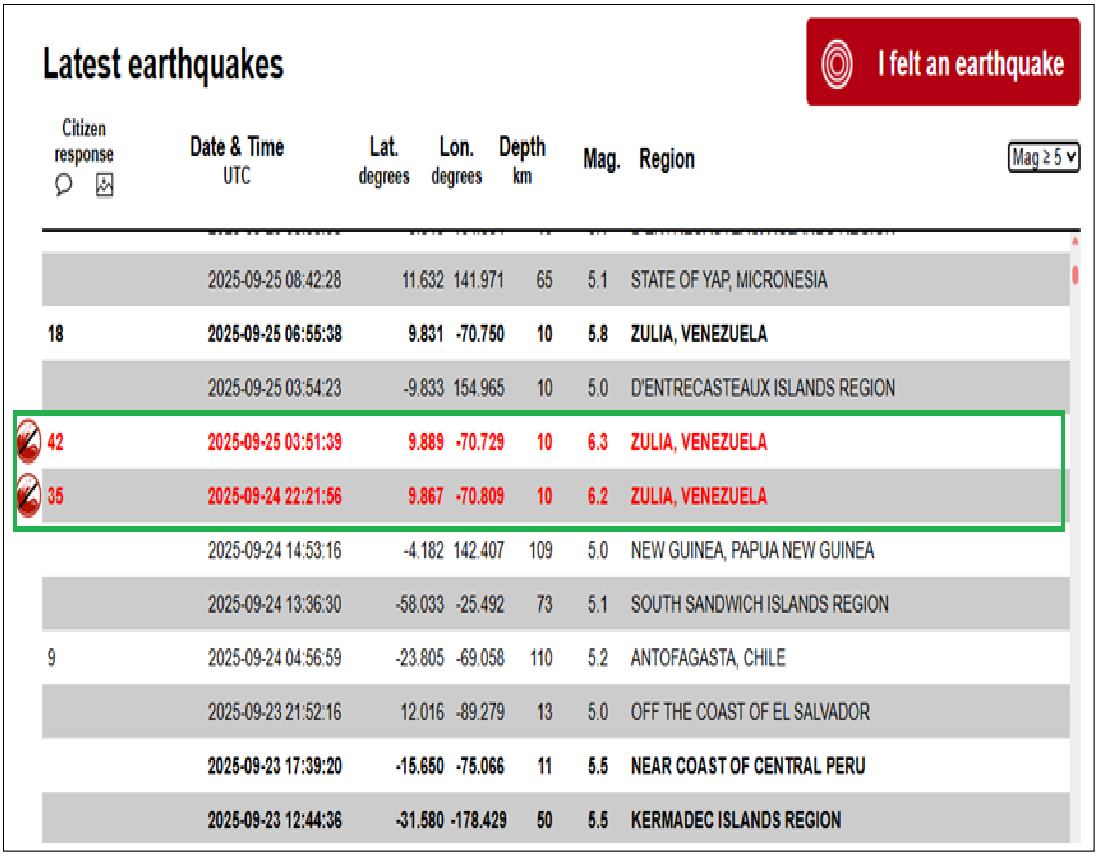


FIGURE B.1: scheli graph from LUNA production server

The locations and magnitudes of the two events were very similar, resulting in nearly identical waveform characteristics.



Latest earthquakes

I felt an earthquake

Citizen response	Date & Time UTC	Lat. degrees	Lon. degrees	Depth km	Mag.	Region
	2025-09-25 08:42:28	11.632	141.971	65	5.1	STATE OF YAP, MICRONESIA
18	**2025-09-25 06:55:38**	**9.831**	**-70.750**	**10**	**5.8**	**ZULIA, VENEZUELA**
	2025-09-25 03:54:23	-9.833	154.965	10	5.0	D'ENTRECASTEAUX ISLANDS REGION
42	**2025-09-25 03:51:39**	**9.889**	**-70.729**	**10**	**6.3**	**ZULIA, VENEZUELA**
35	**2025-09-24 22:21:56**	**9.867**	**-70.809**	**10**	**6.2**	**ZULIA, VENEZUELA**
	2025-09-24 14:53:16	-4.182	142.407	109	5.0	NEW GUINEA, PAPUA NEW GUINEA
	2025-09-24 13:36:30	-58.033	-25.492	73	5.1	SOUTH SANDWICH ISLANDS REGION
9	2025-09-24 04:56:59	-23.805	-69.058	110	5.2	ANTOFAGASTA, CHILE
	2025-09-23 21:52:16	12.016	-89.279	13	5.0	OFF THE COAST OF EL SALVADOR
	2025-09-23 17:39:20	**-15.650**	**-75.066**	**11**	**5.5**	**NEAR COAST OF CENTRAL PERU**
	2025-09-23 12:44:36	**-31.580**	**-178.429**	**50**	**5.5**	**KERMADEC ISLANDS REGION**

FIGURE B.2: Real Earthquake Event

As highlighted in the green boxes (Figure B.1), the initial arrivals, marked by the green circles, correspond to compressional (P-) waves travelling directly through the Earth. These are followed by longer-duration oscillations within the same boxes, which represent surface waves propagating across the Earth's surface. The dispersive nature of surface waves explains the extended arrival times observed in the records. Due to the distance between the seismic source and the recording station, the signals appear with a slight delay, and the sinusoidal waveforms become more pronounced after the initial onset. The latter event, beginning shortly after 04:04 UTC, is of particular interest, as it is further examined through a corresponding `scrttv` plot in the following section.

B.2 Scrttv Plot

The `scrttv` tool provides a real-time display of seismic waveforms across multiple channels, supporting continuous monitoring of incoming data streams. Each trace corresponds to a channel (e.g., HHE, HHN, HHZ), with amplitude variations reflecting ground motion over time. As shown in Figure B.3, the `scrttv` plot presents another view of the same earthquake event recorded at approximately 04:30 UTC. The signal amplitudes across the three channels clearly illustrate the arrival and propagation of seismic waves, complementing the broader temporal perspective provided by the `scheli` plot.

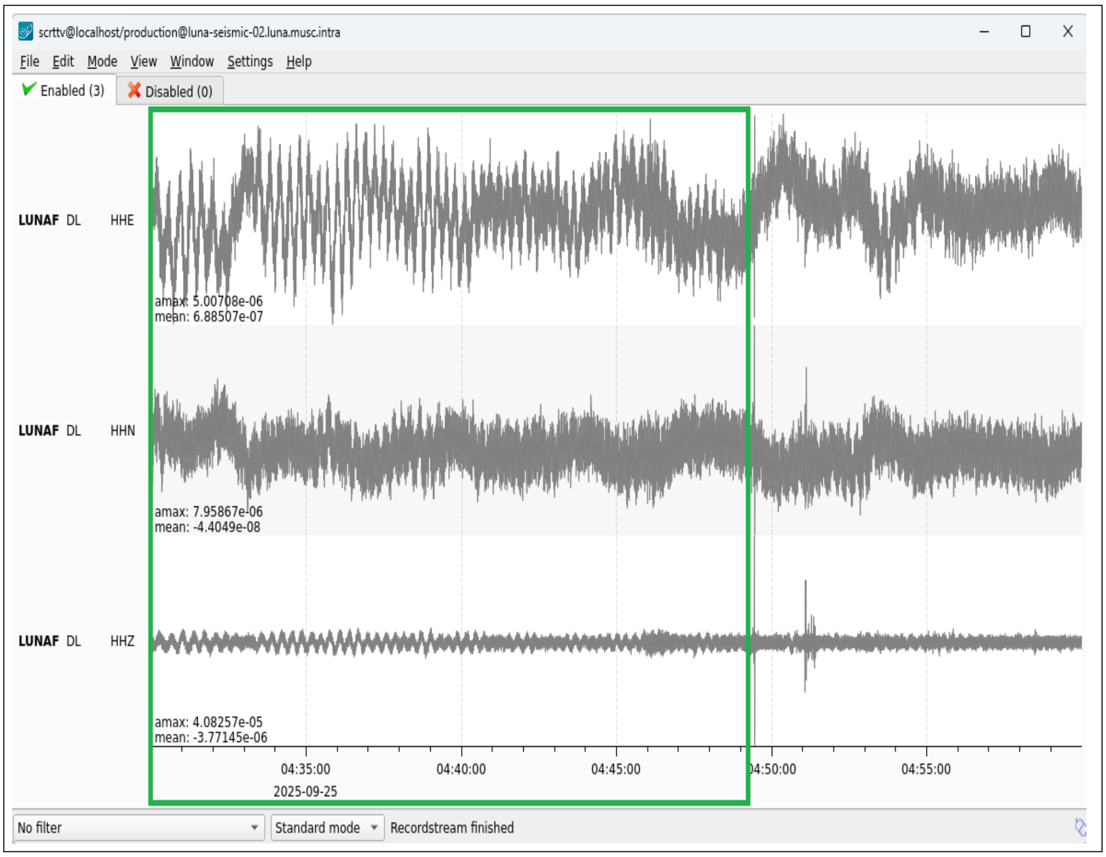


FIGURE B.3: scrttv graph from LUNA production server

References

[1] *The German Aerospace Center (DLR)*. URL: https://www.dlr.de/en (visited on 07/04/2025).

[2] *Microgravity User Support Center (MUSC)*. URL: https://www.dlr.de/en/rb/research-operation/infrastructure/microgravity-user-support-center-musc (visited on 07/04/2025).

[3] *LUNA Analog Facility*. URL: https://www.dlr.de/en/rb/research-operation/infrastructure/luna-analog-facility (visited on 07/04/2025).

[4] *Introduction and Scope — SeisComP Release Documentation*. URL: https://www.seiscomp.de/doc/base/introduction.html (visited on 07/21/2025).

[5] *Seedlink — SeisComP Release Documentation*. URL: https://docs.gempa.de/seiscomp/current/apps/seedlink.html#protocol (visited on 07/21/2025).

[6] *Overview — SeisComP Release Documentation*. URL: https://www.seiscomp.de/doc/base/overview.html (visited on 07/21/2025).

[7] *Scmaster — SeisComP Release Documentation*. URL: https://www.seiscomp.de/doc/apps/scmaster.html (visited on 07/21/2025).

[8] *Slarchive — SeisComP Release Documentation*. URL: https://www.seiscomp.de/doc/apps/slarchive.html (visited on 07/21/2025).

[9] *Scheli — SeisComP Release Documentation*. URL: https://www.seiscomp.de/doc/apps/scheli.html#scheli (visited on 07/21/2025).

[10] *Scrttv — SeisComP Release Documentation*. URL: https://www.seiscomp.de/doc/apps/scrttv.html#scrttv (visited on 07/21/2025).

[11] *Inode(7) - Linux Manual Page*. URL: https://man7.org/linux/man-pages/man7/inode.7.html (visited on 09/23/2025).

[12] *Chapter 10. Managing Users and Groups | Configuring Basic System Settings | Red Hat Enterprise Linux | 8 | Red Hat Documentation*. URL: https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/8/html/configuring_basic_system_settings/managing-users-and-groups_configuring-basic-system-settings (visited on 07/27/2025).

[13] Peter Adams. *Linux Command Line Made Easy*. English. ASIN: B0BKHZFYCM. Independently published, 2022. ISBN: 979-8713654764.

[14] *File Permissions and Attributes - ArchWiki*. URL: https://wiki.archlinux.org/title/File_permissions_and_attributes#Changing_permissions (visited on 07/27/2025).

[15] amrita.sakthivel@suse.com. *Setting Up a Systemd Service*. URL: https://documentation.suse.com/smart/systems-management/html/systemd-setting-up-service/index.html (visited on 07/27/2025).

[16] Freedesktop.org. *systemd.unit - systemd unit configuration*. Accessed: 2025-07-21. 2024. URL: https://man7.org/linux/man-pages/man5/systemd.unit.5.html.

[17] *Understanding Systemd Units and Unit Files | DigitalOcean*. URL: https://www.digitalocean.com/community/tutorials/understanding-systemd-units-and-unit-files (visited on 07/27/2025).

[18] *What Is Bash Scripting? Tutorial and Tips | NinjaOne*. URL: https://www.ninjaone.com/blog/what-is-bash-scripting/ (visited on 07/27/2025).

[19] *Bash Scripting Tutorial – Linux Shell Script and Command Line for Beginners*. freeCodeCamp.org. URL: https://www.freecodecamp.org/news/bash-scripting-tutorial-linux-shell-script-and-command-line-for-beginners/ (visited on 07/27/2025).

[20] *Polkit - ArchWiki*. URL: https://wiki.archlinux.org/title/Polkit (visited on 07/27/2025).

[21] heise online. *Linux-Rechteverwaltung: Mit Sudo Und Polkit Gezielt Root-Rechte Verteilen*. c't Magazin. Aug. 27, 2020. URL: https://www.heise.de/hintergrund/Linux-Rechteverwaltung-Mit-Sudo-und-Polkit-gezielt-Root-Rechte-verteilen-4877932.html (visited on 08/24/2025).

[22] *Chapter 4. PolicyKit | Desktop Migration and Administration Guide | Red Hat Enterprise Linux | 7 | Red Hat Documentation*. URL: https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/7/html/desktop_migration_and_administration_guide/policykit (visited on 07/27/2025).

[23] *PolicyKit - Debian Wiki*. URL: https://wiki.debian.org/PolicyKit (visited on 08/24/2025).

[24] *How to Customize Linux User Environments*. URL: https://www.redhat.com/en/blog/customize-user-environments (visited on 07/27/2025).

[25] *Introduction to Ansible — Ansible Community Documentation*. URL: https://docs.ansible.com/ansible/latest/getting_started/introduction.html (visited on 07/28/2025).

[26] *Learning Ansible Basics*. URL: https://www.redhat.com/en/topics/automation/learning-ansible-tutorial (visited on 07/28/2025).

[27] *What Is Ansible? A Complete Guide to Automation & Deployment*. Scale Computing. URL: https://www.scalecomputing.com/resources/what-is-ansible (visited on 07/28/2025).

[28] *Sample Ansible Setup — Ansible Community Documentation*. URL: https://docs.ansible.com/ansible/latest/tips_tricks/sample_setup.html (visited on 07/28/2025).

[29] *Ansible Playbooks — Ansible Community Documentation*. URL: https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html (visited on 07/28/2025).

[30] *Basic Concepts — Ansible Community Documentation*. URL: https://docs.ansible.com/ansible/latest/network/getting_started/basic_concepts.html#roles (visited on 07/28/2025).

[31] *Introduction to Modules — Ansible Community Documentation*. URL: https://docs.ansible.com/ansible/latest/module_plugin_guide/modules_intro.html (visited on 07/28/2025).

[32] *Ansible.Builtin.Apt Module – Manages Apt-Packages — Ansible Community Documentation*. URL: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/apt_module.html (visited on 08/24/2025).

[33] *Ansible.Builtin.Copy Module – Copy Files to Remote Locations — Ansible Community Documentation*. URL: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/copy_module.html (visited on 08/24/2025).

[34] *Ansible.Builtin.Template Module – Template a File out to a Target Host — Ansible Community Documentation*. URL: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/template_module.html#synopsis (visited on 08/24/2025).

[35] *Ansible.Builtin.Systemd_service Module – Manage Systemd Units — Ansible Community Documentation*. URL: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/systemd_service_module.html#ansible-collections-ansible-builtin-systemd-service-module (visited on 08/24/2025).

[36] *Ansible.Builtin.User Module – Manage User Accounts — Ansible Community Documentation*. URL: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/user_module.html (visited on 08/24/2025).

[37] *Roles — Ansible Community Documentation*. URL: https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_reuse_roles.html (visited on 07/28/2025).

[38] *Ansible Vault — Ansible Documentation*. URL: https://docs.ansible.com/ansible/2.8/user_guide/vault.html#ansible-vault (visited on 07/28/2025).

[39] *Mariadb-Secure-Installation | MariaDB Documentation*. July 10, 2025. URL: https://mariadb.com/docs/server/clients-and-utilities/deployment-tools/mariadb-secure-installation (visited on 07/28/2025).

Declaration of Authorship

I hereby declare that this research document was independently composed and authored by myself.

All content and ideas drawn directly or indirectly from external sources are indicated as such. All sources and materials that have been used are referred to in this project.

The research has not been submitted to any other examining body and has not been published.

Place, date

Signed: Subrata Roy