FOSSEE Winter Internship Report

on

System Administration

Submitted By:

Raghavjit Singh

Guru Nanak Dev Engineering College

Under the Guidance of:

Lee Thomas Stephen

January 2025

Acknowledgment

I want to take this opportunity to express my heartfelt gratitude to everyone who played a crucial role in making my summer internship with FOSSEE - System Administration a valuable and rewarding experience.

First and foremost, I extend my deepest thanks to Mr. Lee Thomas Stephen for believing in my abilities and selecting me for this project. Their unwavering support and encouragement have been instrumental in my progress.

I am deeply grateful to my mentors, whose guidance and wisdom have been indispensable throughout this journey. Their expert advice and timely assistance helped me complete my tasks and enhanced my technical and non-technical skills.

Additionally, I want to express my appreciation to my colleagues who are working on various projects. Their constructive feedback and insightful discussions have significantly contributed to my learning and growth.

I extend my sincerest gratitude to everyone who supported me during this internship for their contributions to this unforgettable experience.

Declaration

I hereby declare that this written submission represents my ideas in my own words. Whenever the ideas or words of others have been included, I have appropriately cited and referenced the sources.

I have accurately acknowledged all sources used in producing this report. Furthermore, I confirm that I have adhered to all academic honesty and integrity principles. I have not misrepresented, fabricated, or falsified any idea, data, fact, or source in this submission.

I understand that any violation of the above principles will lead to disciplinary action by the Institute and may also result in legal consequences from sources that have not been appropriately cited or from those from whom proper permission was not obtained where required.

Raghavjit

Contents

0.1 Introduction

During the tenure of my internship, I had the opportunity to work on several tasks recommended by Mr. Thomas Stephen Lee. These tasks primarily revolved around system administration, including migrating technologies and containerizing applications. This experience allowed me to learn new technologies and provided me with the chance to contribute to real-world projects.

One of the key tasks I completed was migrating virtual machines from VirtualBox to Proxmox using OVA files. The goal was to successfully import these virtual machines into Proxmox and ensure their seamless operation. To simplify the migration process, I developed a script that automated the steps, handled potential failures, and abstracted Proxmox commands for user convenience.

Another significant task involved containerizing the Django-based website, PyFOSS. This website, built using the Django framework, was containerized using Podman. Podman was chosen for its daemonless architecture and enhanced security features. This task improved my understanding of containerization tools and provided hands-on experience with modern application deployment practices.

Additionally, I worked on configuring virtual machines on the ‘yamuna.fossee.in‘ server using VirtualBox. This task aimed to make these virtual machines globally accessible using IPv6. While this task was partially completed, it introduced me to the complexities of managing and configuring network settings for virtual environments.

These tasks honed my technical skills and provided a platform to apply theoretical knowledge to practical challenges.

Chapter 1

VM Migration from VirtualBox to Proxmox

1.1 Task Overview

The task focused on migrating VirtualBox virtual machines (VMs) to Proxmox using OVA files. The main objective was to import these VMs into Proxmox and ensure they boot up successfully. A custom automation script was developed to simplify the migration process, handling errors gracefully and abstracting Proxmox commands for user convenience.

1.2 Task Requirements

1.2.1 OVA File Importation

The task required the provided OVA files to be imported into Proxmox.

1.2.2 Ensuring VM Boot-Up

Post-migration, the VMs had to boot correctly without errors.

1.2.3 Automation Script Development

The script development involved:

- Automating the entire migration process.

- Making the script user-friendly and capable of handling failures.
- Prompting the user for input to simplify Proxmox commands.

1.3 Logs for Task Duration

The migration process was documented in daily logs, detailing the progress and any issues encountered. The logs spanned across nine days of activity:

- Day 1 to Day 9: Detailed information on each day's progress.

1.4 Automation Script Documentation

The automation script was designed to streamline the migration process by automating critical steps, providing user prompts, and ensuring error handling.

1.4.1 Requirements and Limitations

- The script is written in Bash and may not function as expected in other shells.
- Required tools: `fzf`, `awk`, `cut`, `tail`.
- Avoid running multiple instances of the script simultaneously.
- Do not make changes via the Proxmox web interface while running the script.

1.4.2 Usage

```
./migrate [-v] <file>
```

- `-v`: Enables verbose mode for detailed output.
- `<file>`: Path to the OVA or QCOW2 file.

Example:

```
./migrate -v myvm.ova
```

Link to the migrate script: GitHub - Migrate Script

1.5 Process Details

This Bash script automates managing virtual machines on Proxmox, including extracting OVA files, converting VMDK files to QCOW2 format, creating storage, and configuring a new virtual machine. It handles user inputs interactively, checks for proper file usage, and provides options for selecting or creating storage. The script ensures sequential execution of tasks and gracefully cleans up resources in case of failure, with verbose logging for debugging when enabled.

1.5.1 OVA Extraction

Purpose: Extract the contents of the OVA file, which typically contains virtual disk files (e.g., VMDK). This step is essential because the OVA file is a compressed archive, and its contents must be unpacked before proceeding.

Command:

```
tar -xf <ova-file> -C <target-folder>
```

After extraction, locate the virtual disk file (e.g., `.vmdk`) in the target folder for conversion.

1.5.2 Disk Conversion to QCOW2

Purpose: Convert VMDK file (input-vmdk) to QCOW2 format, which Proxmox prefers due to its advanced features like snapshot support and efficient storage usage. This step ensures the disk file is compatible with Proxmox. This can be done using qemu-img.

Command:

```
qemu-img convert -f vmdk -O qcow2 <input-vmdk> <output-qcow2>
```

The resulting QCOW2 file will be used as the primary disk for the VM in Proxmox.

1.5.3 Storage Management

Purpose: Prepare a suitable storage location in Proxmox to host the virtual disk. This step is critical before attaching the QCOW2 (obtained in the previous step) file to a virtual machine. If storage already exists, this step can be skipped; otherwise, create new storage using the command below.

Command to Create Storage:

```
pvesm add lvmthin <storage-name> --vgname <vg-name> --thinpool <thinpool-name> --content images
```

Once the storage is ready, note its name and ensure it is accessible for the subsequent steps.

1.5.4 VM Creation

Purpose: Create a virtual machine in Proxmox to utilize the converted QCOW2 disk. This step provides the VM with the necessary resources like memory, CPU cores, and network configuration.

Command:

```
qm create <vm-id> --name <vm-name> --memory <memory-size> --cores <cpu-cores> --net0 virtio,bridge=vmbr
--scsihw virtio-scsi-pci --boot order=scsi0
```

This creates the VM definition but does not yet attach the disk. Ensure the VM ID and other parameters match your requirements.

1.5.5 Disk Attachment to VM

Purpose: Attach the converted QCOW2 disk to the created VM. This step links the virtual disk file to the VM so it can be used as the primary or additional storage for the virtual machine.

Command:

```
qm set <vm-id> --scsi0 <storage>:<disk>,size=<size>
```

After attaching the disk, the VM is ready to boot and use the QCOW2 file as its virtual disk.

1.6 Cleanup and Error Handling

The script includes a comprehensive cleanup and error-handling mechanism to ensure smooth execution and failure recovery. Each critical step checks for potential errors and takes appropriate actions. Below is a detailed explanation of the cleanup and error-handling strategies:

1.6.1 Error Handling in Steps

- **Input Validation:** The `getInput` function ensures valid user input, preventing accidental empty values that could disrupt the script's flow. It repeatedly prompts the user until valid input is provided.
- **File Checks:** The `checkUsageAndFile` function verifies the existence of the specified file. If the file is not found or the usage is incorrect, appropriate error messages are displayed, and the script halts further execution.
- **Extraction Validation:** During extracting OVA files, the script checks for existing VMDK files in the target folder. If a VMDK file is found, the user is prompted to decide whether to overwrite it, ensuring no unintended file overwrites.

- **Conversion Errors:** The `convertQCOW` function captures the status of the conversion process. If the conversion from VMDK to QCOW2 fails, an error message is displayed, and the script stops further operations.
- **Storage Creation:** In the `createStorage` function, errors in creating a new storage are detected. The script notifies the user and allows them to retry or choose an alternative storage option.
- **VM Creation and Disk Import:** During the creation of the virtual machine (`createVM`) and disk import (`importQCOW2`), errors are identified, and the script ensures that failed operations are not left in an inconsistent state.
- **Storage Attachment:** Errors in attaching storage to the VM are handled gracefully, ensuring that incorrect configurations are not applied.

1.6.2 Interactive Cleanup Process

When an error occurs during any step, the script invokes the `handleFailure` function to clean up resources. This function is interactive, ensuring that users have control over the cleanup actions:

- **Folder Cleanup:** If a folder was created during the extraction process, the user is prompted to confirm its deletion. If confirmed, the folder and its contents are safely removed.
- **VM Cleanup:** If a virtual machine was partially created, the script stops the VM, detaches any associated storage, and deletes the VM upon user confirmation.
- **Disk Cleanup:** If a virtual disk was created, the script asks the user if it should delete the disk. Upon confirmation, the disk is removed from the storage pool.

1.6.3 Execution Flow

The main script executes all functions sequentially. Any function that encounters an error returns a non-zero status, stopping further execution and invoking the cleanup process. The verbose mode (`-v`) provides detailed feedback at each step, allowing users to understand the operations being performed and the causes of any errors.

1.7 Proxmox CLI Commands

Proxmox's CLI is essential for managing VMs, storage, and disks. Below are some of the commonly used commands:

1.7.1 VM Management Commands

- **Create VM**:

  ```
  qm create <vm-id> [OPTIONS]
  ```

- **Destroy VM**:

  ```
  qm destroy <vm-id>
  ```

- **Start VM**:

  ```
  qm start <vm-id>
  ```

- **Stop VM**:

  ```
  qm stop <vm-id>
  ```

- **Change VM Configuration**:

  ```
  qm set <vm-id> [OPTIONS]
  ```

- **List All VMs**:

  ```
  qm list
  ```

1.7.2 Storage Management Commands

- **List Storages**:

  ```
  pvesm status
  ```

- **Delete Storage**:

  ```
  pvesm remove <storage-name>
  ```

- **Create Storage**:

  ```
  pvesm add <type> <storage-name> [OPTIONS]
  ```

1.8 Conclusion

The task of migrating VirtualBox VMs to Proxmox was completed using a combination of Proxmox CLI tools and the custom automation script. The script was designed to streamline the process, minimize errors, and enhance user experience, ensuring a seamless migration workflow.

Chapter 2

Containerize Django Website 'pyfoss' Using Podman

2.1 Task Overview

The objective of this task was to containerize the `pyfoss` website, which is built using the Django framework. Podman was chosen as the containerization tool due to its daemon-less architecture and enhanced security features.

2.2 Task Requirements

The following resources were used for this task:

- The `pyfoss` website code can be found in the GitHub repository, available *[here]*.
- The database used for the website is `Sqlite3`, which is a lightweight relational database.

2.3 Logs for Task Duration

Detailed logs of the task's progress are available for each day:

- **Day 1**: *[Logs/Day1.md]*
- **Day 2**: *[Logs/Day2.md]*
- **Day 3**: *[Logs/Day3.md]*

2.4 Dockerfile Documentation

To understand the functionality and usage of the Dockerfile, refer to the detailed Dockerfile documentation below.

2.4.1 PyFOSS Dockerfile Documentation

The following documentation provides an overview of the Dockerfile created for containerizing the `pyfoss` application. The setup is optimized to minimize image layers and size while providing flexibility in configuring the application through the `config.py` file and `ALLOWED_HOSTS` setting.

Requirements and Limitations

- The Dockerfile is compatible with both Podman and Docker.
- Efforts were made to minimize the image size and the number of layers for optimal efficiency.
- Users can modify the `config.py` file and `ALLOWED_HOSTS` through build-time arguments (`--build-arg`).
- The container assumes that the dependencies specified in `requirements.txt` are correct and complete. Some minimal code changes were made *[commit-log]*.
- Ensure that Podman or Docker is properly configured before running the build.

Usage

Build the Image: To build the container image, use the following command:

```
podman build --build-arg ALLOWED_HOSTS="192.168.1.100" --build-arg CONFIG_FILE=./my-config.py -t pyfoss
```

- `ALLOWED_HOSTS`: A comma-separated list of allowed hosts for the Django application. Defaults to `127.0.0.1`.
- `CONFIG_FILE`: Path to the custom `config.py` file to be copied into the container. Defaults to `./config.py`.

Run the Container: Once the image is built, run the container with the following command:

```
podman run -d -p 8000:8000 --name pyfoss-app pyfoss-app
```

This command maps the container's port `8000` to the host's port `8000`.

Process Details

The Dockerfile is designed to create a containerized environment for the PyFOSS application. It uses `quay.io/roboxes/alma8` as the base image, ensuring compatibility with the application. Key steps include installing necessary dependencies, setting up a Python virtual environment, and configuring the application dynamically to accommodate changes like `ALLOWED_HOSTS` and custom configuration files. Finally, the Dockerfile prepares the application to run using Django's development server.

1. Base Image Setup: The base image used is `quay.io/roboxes/alma8`, which offers a lightweight and stable platform for the PyFOSS application. The Dockerfile starts by specifying this base image:

```
FROM quay.io/roboxes/alma8
```

2. Dependency Installation: Essential Python development tools and libraries are installed to meet the application's requirements, including packages for MySQL integration:

```
RUN dnf install -y python3-devel pkgconfig git kernel-headers mysql-devel
```

3. Application Setup: The PyFOSS application is cloned from the GitHub repository, and a Python virtual environment is set up for dependency installation:

```
RUN git clone https://github.com/Raghavjit/pyfoss && \
    cd pyfoss && \
    python3 -m venv venv && \
    source venv/bin/activate && \
    pip install --no-cache-dir -r requirements.txt
```

4. Dynamic Configuration: This step allows customization of `ALLOWED_HOSTS` and the inclusion of a custom `config.py` file, without modifying the Dockerfile itself:

```
ARG ALLOWED_HOSTS="127.0.0.1"
ARG CONFIG_FILE=./config.py
COPY ${CONFIG_FILE} /home/pyfossuser/pyfoss/pyfoss/config.py
RUN sed -i "/ALLOWED_HOSTS = \[/ s/\]$/'${ALLOWED_HOSTS}']/g" pyfoss/settings.py
```

5. Application Launch: Finally, the application is launched by running Django's development server, making the application accessible at `0.0.0.0:8000`:

```
CMD ["sh", "-c", "source venv/bin/activate && \
                  python3 manage.py makemigrations && \
                  python3 manage.py migrate && \
                  python3 manage.py runserver 0.0.0.0:8000"]
```

Notes

- **Custom Configuration:** Users can provide a path to a custom `config.py` using the `CONFIG_FILE` argument, and modify `ALLOWED_HOSTS` through the `--build-arg ALLOWED_HOSTS` flag.
- **Optimized Image:** The Dockerfile minimizes the number of layers by combining related commands and using `--no-cache-dir` with `pip install` to reduce image size.
- **Debugging:** For interactive troubleshooting, use the `-it` flag with `podman run`, as shown below:

```
podman run -it --rm pyfoss-app Bash
```

2.5 Conclusion

The task of containerizing the PyFOSS website using Podman was completed. The Dockerfile created for this task encapsulates all the necessary dependencies for the website to function. Efforts were made to ensure that the final container was as minimal and efficient as possible, and the application is now ready to run in a containerized environment.

Chapter 3

Assigning IPv6 Static to VirtualBox VM

3.1 Task Overview

The task involved creating virtual machines on the `yamuna.fossee.in` server using VirtualBox and making these VMs globally accessible using IPv6.

3.2 Task Requirements

1. Hosting services on IPv6 should be possible.
2. Securely SSH into the server using IPv6.
3. Apply standard FOSSEE `firewalld` rules for IP whitelisting.

3.3 Assigning IPv6 to VM

3.3.1 Creating the VM

The first step was to create a VM on a local machine where RHEL could be installed. After the VM was created, the following points were noted:

- Set a password for SSH login.
- Disable hardware nesting and nested paging.

- Ensure the firewall is turned off.

3.3.2 Importing the VM

To import the machine, the following command was used:

```
vbm import RHEL.ova --vsys 0 --name "RHEL_1"
```

3.3.3 Assigning an IP to the VM

Its IP was needed to SSH into the VM. The method used was to create a host-only network (`vmbr0`) connection and then obtain the IP to SSH.

```
vbm hostonlyif create
```

This created a `vboxnet0` host-only network with the subnet `192.168.56.0/24`. Then:

```
vbm modifyvm <VM-NAME> --nic<interface number> hostonly --hostonlyadapter<interface number> <interface
vbm modifyvm RHEL_1 --nic1 hostonly --hostonlyadapter1 vmbr0
```

To obtain the IP, the following methods were used:

```
nmap -sn 192.168.56.0/24
# or
cat ~/.config/VirtualBox/HostInterfaceNetworking-vboxnet0-Dhcpd.leases
```

Note: VirtualBox resolves DHCP only on the first interface; all other interfaces must be manually activated using `dhclient`.

```
dhclient <interface name>
# or
dhclient -6 <interface name>
```

3.3.4 Creating a Bridged Network

A bridged interface was created to assign an IPv6 to the VM (via DHCP). This interface made the VM behave like it was connected to the same DHCP server as the host machine.

```
vbm modifyvm <VM-NAME> --nic<interface number> bridged --bridgeadapter<interface number> <host interfac
vbm modifyvm RHEL_1 --nic2 bridged --bridgeadapter2 enp3efdo9
```

Note: The bridged connection was configured on `nic2` and not `nic1`. The `nic1` in host-only mode obtained an IP, enabling SSH access to the VM.

3.3.5 Assigning IPv6

Since the bridged connection was on `nic2`, it did not automatically obtain an IP. An IPv6 address was manually assigned. The host (`yamuna.fossee.in`) had IPv6 `2a01:4f9:3080:2d4d::2`, so an IP in the same subnet was assigned to the VM (e.g., `2a01:4f9:3080:2d4d::5`). This was done using either `nmtui` (recommended) or the `ip` command:

```
sudo ip -6 addr add 2a01:4f9:3080:2d4d::5/64 dev enp0s8


sudo ip -6 route add default via 2a01:4f9:3080:2d4d::2 dev enp0s8


echo "nameserver 2001:4860:4860::8888" | sudo tee -a /etc/resolv.conf
echo "nameserver 2606:4700:4700::1111" | sudo tee -a /etc/resolv.conf
```

3.3.6 Enabling Internet Access

Although the VM now had a globally accessible IPv6 address, many services (e.g., `dnf`) did not support IPv6. Therefore, an IPv4 was assigned for internet access:

```
vbm controlvm RHEL_1 acpipowerbutton
vbm modifyvm RHEL_1 --nic1 NAT
```

Note: NAT was configured on `nic1` for proper functioning.

3.4 IP Whitelisting

3.4.1 Firewall Rules Applied

FOSSEE uses `firewalld` as its firewall. The following rules were applied to the host machine:

- **Zone-wise Rules:**
  - FOSSEE
  - IITB
  - SysAd
  - Team (Modified to allow access to me)
  - VM

3.4.2 Problems Faced

Pinging

Upon activating these rules, network packets could no longer reach the VM's IPv6 `2a01:4f9:3080:2d4d::5`. The ping result:

```
PING 2a01:4f9:3080:2d4d::5 (2a01:4f9:3080:2d4d::5) 56 data bytes
From 2a01:4f9:3080:2d4d::2 icmp_seq=1 Destination unreachable: Administratively prohibited
...
```

SSH

SSH was also not working, either because port 22 was closed or the protocol was blocked:

```
ssh: connect to host 2a01:4f9:3080:2d4d::5 port 22: Permission denied
```

Traceroute

Traceroute investigation showed:

```
traceroute to 2a01:4f9:3080:2d4d::5 (2a01:4f9:3080:2d4d::5), 30 hops max, 80 byte packets
...
12  ae15-0.fra20.core-backbone.com (2a01:4a0:1338:166::1)
...
```

3.5 Conclusion

The task involved setting up IPv6 addresses for VMs and ensuring seamless connectivity while adhering to FOSSEE security standards. While the IPv6 configuration succeeded, firewall rules interfered with network access. Recommendations include:

- Use **KVM/QEMU**: Easy to use and more powerful command-line tools.
- Use **Proxmox**: Offers a convenient WebUI and globally accessible VMs.

Bibliography

1. RaghavJit. *Daily Progress: Documentation and Logs for Internship.* GitHub Repository. Available at: `https://github.com/RaghavJit/DailyProgress`.

2. The GNU Project. *GNU Awk User's Guide.* Available at: `https://www.gnu.org/software/gawk/manual/gawk.html`.

3. Junegunn Choi. *fzf: A Command-Line Fuzzy Finder.* GitHub Repository. Available at: `https://github.com/junegunn/fzf/blob/master/README.md`.

4. QEMU Disk Image. *qemu-img: qemu-img allows you to create, convert, and modify images offline. It can handle all image formats supported by QEMU..* Available at: `https://qemu-project.gitlab.io/qemu/tools/qemu-img.html`.

5. Proxmox. *proxmox: a platform to run virtual machines and containers..* Available at: `https://pve.proxmox.com/pve-docs/pve-admin-guide.html`.

6. VirtualBox *vboxmanage: command line tool to use Virtualbox..* Avalible at: `https://www.virtualbox.org/manual/ch08.html`

7. Firewalld Project. *firewall-cmd: Man Page.* Available at: `https://firewalld.org/documentation/man-pages/firewall-cmd.html`.