jamk

Implementing a Software Update Tool for a Laboratory Automation System

Kalle Lehikoinen

Bachelor's thesis
May 2024
Bachelor's Degree Programme in Information and Communications Technology

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Description

Lehikoinen, Kalle

Implementing a Software Update Tool for Laboratory Automation System

Jyväskylä: Jamk University of Applied Sciences, May 2024, 30 pages

Degree Programme in Information and Communications Technology, Bachelor's thesis.

Permission for open access publication: Yes

Language of publication: English

Abstract

This bachelor's thesis documents the initial development of the System Remote Access (SRA), a configuration tool for laboratory automation system. The thesis was commissioned by Thermo Fisher Scientific, located in Vantaa, a company specializing in clinical chemistry analyzers, industrial solutions, and laboratory automation.

The objectives of this thesis were to kickstart the development of the SRA tool and implement its first operational feature, which initiates software updating process within laboratory automation system. The thesis was scoped to implement seamless software update process in one automation system unit, Laboratory Automation Controller.

The project included configuring support for seamless software updates in the controller PC, developing a graphical user interface to initiate the software update process and implementing the backend software needed to manage the update process in the controller unit.

The controller unit was successfully updated, only downtime being the reboot phase of the device. The results indicate that the initial implementation of the feature, while functional, offers a base for future refinement. In a more refined model, the update process could be more error-proof, more structured and rely more on system level processes.

Keywords/tags (subjects)

Linux, Laboratory automation, Software update

Miscellaneous (Confidential information)

jamk

Description

Lehikoinen, Kalle

Ohjelmistopäivitystyökalun toteuttaminen laboratorioautomaatiojärjestelmään

Jyväskylä: Jyväskylän ammattikorkeakoulu. Toukokuu, 2024, 30 sivua

Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: englanti

Julkaisulupa avoimessa verkossa: kyllä

Tiivistelmä

Tämä opinnäytetyö dokumentoi laboratorioautomaatiojärjestelmän konfigurointityökalun (SRA) ensimmäiset kehitysvaiheet. Opinnäytetyön toimeksiantajana toimii Thermo Fisher Scientic, jonka Vantaan toimipisteen erikoisalaan kuuluu kliiniset kemian analysaattorit, teollisuusratkaisut ja laboratorioautomaatio.

Opinnäytetyön tavoitteena oli käynnistää SRA-työkalun kehittäminen ja toteuttaa sen ensimmäinen toiminnallinen ominaisuus, joka mahdollistaa ohjelmistopäivitysten toteuttamisen laboratorioautomaatiojärjestelmässä. Opinnäytetyö rajattiin saumattoman ohjelmistopäivityksen toteuttamiseen laboratorioautomaatiojärjestelmän ohjausyksikössä.

Projekti piti sisällään saumattomien ohjelmistopäivitysten mahdollistamisen ohjausyksikössä, graafisen käyttöliittymän kehittäminen ohjelmistopäivitysten aloittamiseksi sekä päivitysohjelmiston toiminnallisuuden kehittämisen.

Projektin tuloksena ohjausyksikkö päivitettiin onnistuneesti, ainoa käyttökatko syntyi laitteen uudelleenkäynnistyksestä. Tulokset osoittivat, että ominaisuuden ensimmäinen toteutus, vaikka olikin käyttökelpoinen, jättää tilaa parannuksille tulevaisuudessa. Jatkokehitetyssä mallissa päivitysprosessi voisi olla virheenkestävämpi, hallittavampi ja perustua enemmän järjestelmätason prosesseihin.

Avainsanat (asiasanat)

Linux, Laboratorioautomaatio, Ohjelmistopäivitys,

Muut tiedot (salassa pidettävät liitteet)

Contents

Figures

Tables

Definitions

Bash	Bourne-Again Shell, a shell program and command language
BIOS	Basic Input-Output System
CPU	Central Processing Unit
ext4	Fourth Extended filesystem, a native Linux filesystem type
fstab	File System Table, a configuration file that defines how disk partitions are mounted
GUI	Graphical User Interface
JSON	JavaScript Object Notation, a lightweight data-transfer format
LAC	Laboratory Automation Controller
RAM	Random-Access Memory
SRA	System Remote Access, a configuration tool for laboratory automation system
UEFI	Unified Extensible Firmware Interface
UUID	Universally Unique Identifier

1 Introduction

1.1 Thermo Fisher Scientific

The thesis assignment was commissioned by Thermo Fisher Scientific, which is an American supplier of analytical instruments, life science solutions, and specialty diagnostics. In Vantaa, Finland the company employs approximately 600 employees and specializes in clinical chemistry analyzers, industrial solutions and laboratory automation. (Thermo Fisher Scientific, 2023).

1.2 Thesis assignment

In laboratory operations, minimizing downtime during maintenance is essential. According to survey by Agilent Technologies (2017), downtime during laboratory maintenance, whether planned or unplanned, is seen as major operational challenge among laboratory employees. Therefore, ability to install software updates without interrupting laboratory functionality is crucial in environments that require high availability.

The project assignment involves launching the development of the System Remote Access (SRA), which is a multi-purpose tool to monitor and configure laboratory automation system, as well as uploading software updates to the system. Updating software is the first feature of SRA and the subject of this thesis. In later phases SRA can be used to multiple administrative tasks, such as downloading log files, editing system configuration, and requesting system diagnostics.

The first objective is to implement the software foundation of SRA, in such way that it is scalable, as more features will be implemented in the future. The second objective of the project is to implement a feature that enables seamless software update process in the automation system. The project will be carried out in a cross-functional Scrum team at Thermo Fisher Scientific Vantaa.

1.3 Scope of the thesis

This thesis focuses on the implementation of the software updating feature in Laboratory Automation Controller (LAC) PC, a unit that manages the automation system and serves as a starting point for system-wide software updates. This focus was chosen to initially explore the methods and technologies that is needed to install software update without disrupting ongoing operations.

Given that the SRA tool operates on this same unit, choosing LAC PC for software updates in this initial iteration is a logical decision. The scoped objective for this thesis is the initial implementation of the core software for the SRA tool, following with the implementation of software updating feature in the controller PC. The first part of this thesis research technologies and technical requirements to implement the feature and the second part focuses on developing the tool with the software updating feature.

2 Theory

2.1 Overview

The theoretical part of this thesis, research the technologies and methods relevant to implementing seamless software updates. The theory can be divided into three main components:

- Linux operating system
- Process of seamless software update
- Application frameworks and technologies

For the operating system component, information was gathered through searches from Jyväskylä University of Applied Sciences library's search interface Janet Finna. The sources on seamless software updates were found from official documentations, technology articles, and blogs from developers that have implemented such systems. For the application frameworks the information was gathered almost exclusively from online documentations released by the developers of each technology, which should guarantee that the information is accurate and up to date.

The software and frameworks described in this theoretical section is selected primarily because they are already used in the company. This choice is rational, as the used technologies are compatible with existing ones and leverages the current knowledge of other developers.

2.2 Scrum

Scrum is an agile framework used in software development to manage projects. Some of the main characteristics in Scrum are iterative delivery style and adaptive planning. The key roles in Scrum are the Product Owner, Scrum Master, and Development Team. The project is usually divided into

two-week iterations, called sprints. Each sprint includes Scrum events such as sprint planning, sprint reviews, and retrospectives. Daily stand-up meetings are held to track progress and coordinate work. (Flewelling, 2018, Chapter 2)

2.3 Linux operating system

Linux is an open-source operating system that is known for customization capabilities and strong security features. It has been popular choice among IT professionals and organizations for several years. The operating system comes with different versions, known as distributions, that are collections of specific kernel and software tailored to meet various needs. Most of Linux distributions are free of charge and very scalable, making Linux a versatile and economical option for businesses. Some of well-known distributions are CentOS, Debian, Red Hat and Ubuntu. (Smith, 2012, p. 17)

2.3.1 Boot process

When a device running Linux operating system is turned on, series of events are triggered that eventually leads to a normally running operating system. Firstly, the firmware starts and performs check that hardware runs properly and then it searches for a bootloader program to run from a bootable device. Secondly, the bootloader program is executed, and it determines what kernel program is loaded. Finally, the kernel is loaded into memory and starts the necessary background tasks for the system to operate normally. (Breshnan & Blum, 2021, Chapter 10)

2.3.2 Architecture

A Linux system can be divided into three main levels (Figure 1). At the lowest level is the hardware that includes all the physical components such as CPU, RAM, Storage space and Network interfaces. In the core of the Linux system is the kernel. The kernel is responsible for managing processes, memory allocation and acts as an interface between the higher-level software and the hardware. (Ward, 2021, Chapter 1.1)

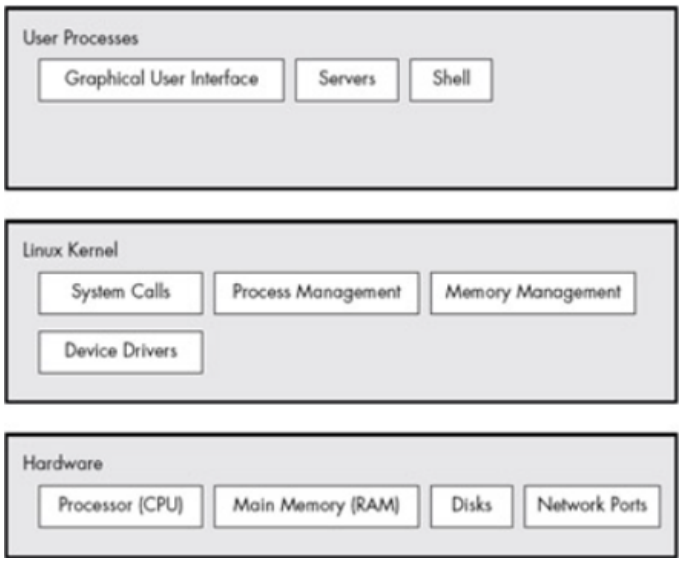

Figure 1. Linux architecture (Ward 2021, Figure 1-1)

2.3.3 Filesystem and partitions

Partitions are defined in a specific area of the disk known as the partition table. Partitions divide a physical storage drive into isolated sections (Figure 2). Partition table and each partition of desired size can be created using command line tools, such as *parted*. After partitions are created, they can be formatted into a filesystem. (Ward 2021, Chapter 4)

A filesystem is all the data, files and directories that user interacts with. As Ward (2021, Chapter 4.2) states: “It supplies the structure to transform a simple block device into the sophisticated hierarchy of files and subdirectories that users can understand.”

To access the data in the filesystem the correct partition location from the partition table needs to be used. Linux filesystems are usually formatted into *ext4* format, which is native to Linux. When filesystem is created, a unique identifier is generated to identify the filesystem. This UUID, for instance, allows the kernel to identify filesystems during the boot process. When the device boots, the kernel program mounts the root filesystem to the running system. (Ward 2021, Chapter 4.2)

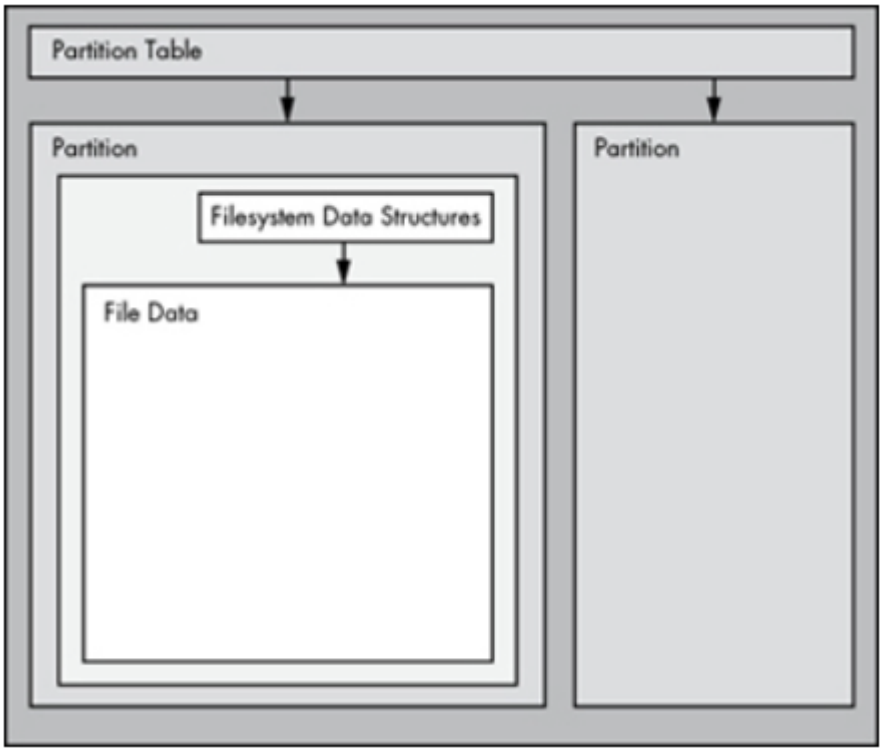

Figure 2. Typical Linux disk structure (Ward 2021, Chapter 4).

2.3.4 GRUB boot loader

Boot loaders and firmware work closely together. Modern Linux systems use nearly universally GRUB, version 2, as a bootloader (Yogesh 2020 Chapter 3; Ward 2021, Chapter 5.4.2). As firmware the UEFI has mostly replaced the older BIOS (Banik 2022, Chapter 3). At the beginning of the boot process the boot loader loads the kernel into memory and runs it. On PCs, boot loaders use firmware to access disks to read the files that is needed to load the kernel.

GRUB has capability to navigate in the filesystem to find the kernel image and select the boot configuration. The GRUB configuration file determines which kernel and filesystem is loaded at startup along with other boot parameters, such as splash image and whether the filesystem should be mounted in read-only mode. Additionally, GRUB allows users to manually select the kernel and the root filesystem through its command line interface, instead of reading settings in the configuration file. (Ward 2021, Chapter 5) If there's an issue during the boot process, usually the user ends up to this command line interface to recover the system.

GRUB can be installed using *grub install* command that copies the boot loader into place and informs the firmware about this boot loader. On systems that have UEFI firmware GRUB is installed by running command:

```
# grub-install --efi-directory=efi_dir --bootloader-id=name
```

In this command *efi_dir* is the path of the UEFI directory in the system, typically mounted to /boot/efi. The second argument is an identifier. If the boot loader is installed in a disk that is used later in another system, the new systems firmware needs to be informed about this boot loader. (Ward, 2021, Chapter 5.5.3)

2.3.5 Bash scripting

Operating systems include a command-line tool, a shell, which enables user to interact with the system by entering commands in a prompt. Through this interface users can launch programs, manage processes, and manipulate files. Bash is the default shell for almost all the Linux distributions, and it runs as a command prompt as demonstrated in Figure 3. (McGrath, 2019)

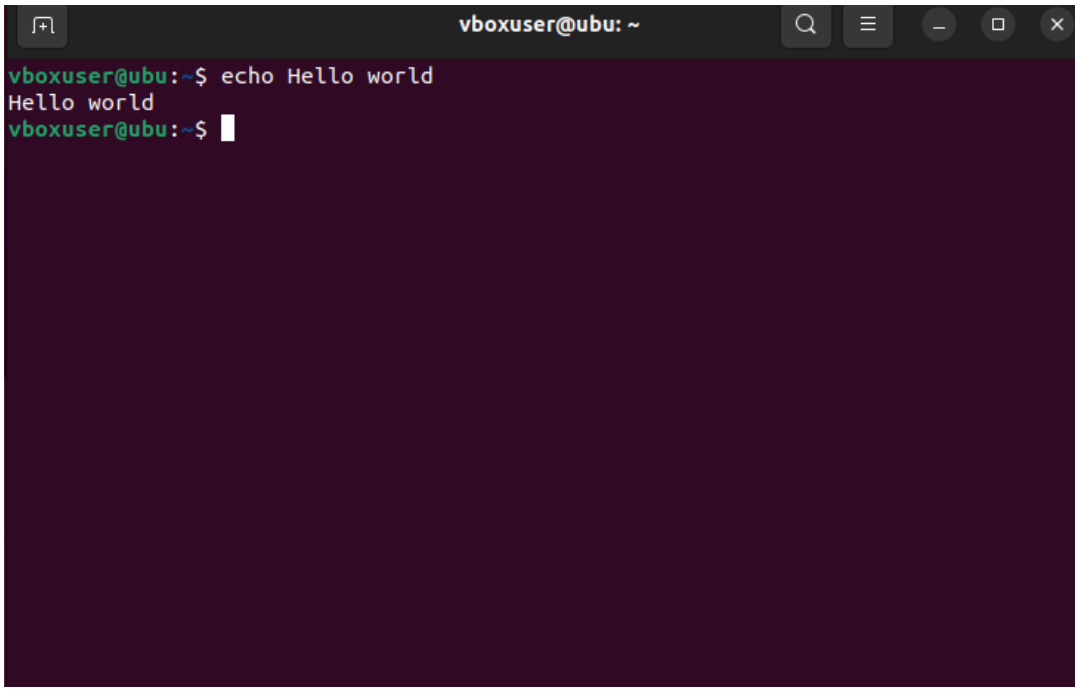

Figure 3. Simple command in Bash shell

Bash can be understood as either a shell or a programming language. As a programming language its core design is to execute other programs in an organized manner. Bash is particularly suitable for automating tasks by running sets of commands that needs to run together. (Ryder, 2018) Figure 4 demonstrates a simple Bash script that takes an argument as input.

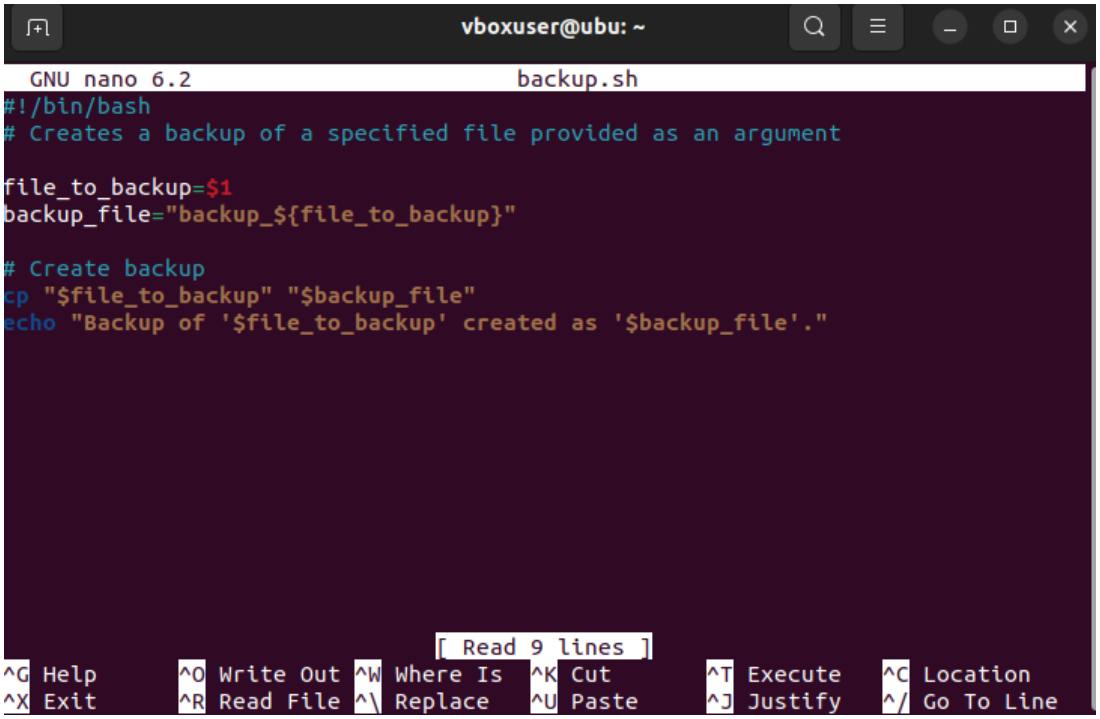

Figure 4. Bash script with an argument

Following Table 1 presents some of commands that are used in Bash scripts in this thesis.

Table 1. Bash commands (DigitalOcean, 2023; Javapoint, N.d; Red Hat, N.d)

Command	Description
mount	Mount a filesystem
umount	Unmount filesystem
mount --bind	Remount part of a file hierarchy in different location, original location still available
blkid	Shows information about block devices

Table 2. (continued)

Command	Description
chroot	Changes root directory to a new directory
dd	Used for copying block devices
echo	Writes to standard output
cp	Copies file

2.4 Seamless software update

Seamless software update, also known as dual boot or A/B system update can be installed without interrupting the device by using dual partitions. This dual partition configuration ensures that one functional system is always available, even if the update to the new partition fails. (Android, 2024)

In the beginning of the update process the system runs in the active system partition (A). When update process is started, the inactive system partition (B) is installed with the update and verified. After successful installation, partition B becomes active, and the bootloader is configured to boot the system from the updated partition. If there is an error with the new update during the reboot, the system can revert to the A partition, allowing the device to function normally as it did before the update. (Pasternak, 2024)

A familiar example that utilizes seamless software update is the Android operating system. As stated in the Android official documentation (2024) the update is downloaded in the background to an inactive partition while the user continues to use the device normally. After the download is finished, the user reboots into the updated partition. Only downtime during an update is when the device reboots. If the update fails, the system reboots using the old partition.

The same A/B update mechanism that is used in mobile phones, can also be implemented in Linux systems. Linux systems can be updated using A/B method, where one partition is actively used while the other is updated (Byrne, 2018). This dual partition approach increases the reliability of

the system during updates and is ideal for devices that require high availability. Therefore, it may be considered a viable strategy to update software in a laboratory automation environment. One important benefit of the A/B update method is that it updates simultaneously all software components. This approach allows the software to be tested and verified to function together before the installation.

2.5 Application frameworks and technologies

2.5.1 Docker

Docker is a platform that makes it possible to run applications in an isolated environment, separated from the host machine. Docker container is runnable package containing all the software and required dependencies, ensuring the container to run in any machine. Containers are built from Docker images, which are templates with instructions for creating the container. By default, containers are isolated from the host machine, but this can be controlled based on use case. One way of configuring the Interaction between host machine and the container is mounting specific parts of host filesystem to the container through volumes. Volumes are used to access and persist data used by Docker containers. Docker Compose defines the application environment with a Docker Compose file, where services, networks, and volumes can be configured. Volumes can be defined in the Docker compose file as well and can be connected to containers, allowing data to persist even after the container itself is deleted. (Docker, N.d)

2.5.2 ASP.NET Boilerplate

ASP.NET Boilerplate (ABP) is an application framework which provides architectural model based on Domain Driven Design. ABP automates common software development tasks such as user, role and permission management. This allows developers to focus more on business logic than boilerplate code. ABP comes with ready-to-use templates that are useful for enterprises and developers looking to launch their project with fully configured environment. It can be integrated with modern frontend frameworks like Vue.js. (.NET foundation, N.d)

2.5.3 Vue.js

Vue.js is a JavaScript framework for building user interfaces and single-page applications. The framework utilizes a component-based architecture that can be used to develop projects of any complexity. It is designed to be flexible, incrementally adoptable and enhances code reusability. (Vue.js, n.d) These characteristics suggest that Vue.js is a good choice for a project that starts with one feature but is intended to scale up over time.

2.5.4 SHA256 checksum

SHA256 algorithm is one of the most used hashing algorithms. It can be used to verify file integrity during file transfer. The hash value, a checksum, is calculated before and after the file transfer and the values are compared. Any change in the data results in a completely different hash value. (Jena, 2023)

3 Implementation

3.1 Software update process

Figure 5 demonstrates the software update process as a “happy path” scenario, including each step of user interaction required to complete the update successfully.

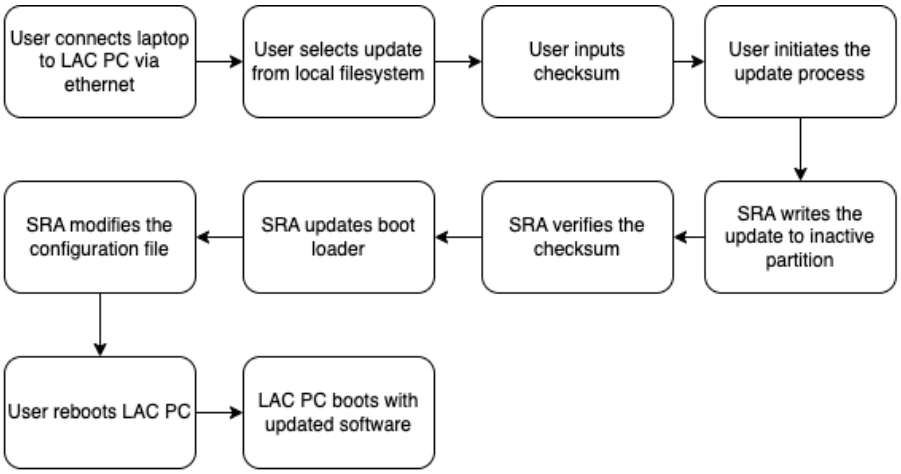

Figure 5. Software update process

3.2 LAC PC

The LAC software is running in a Docker container in the active partition of the LAC PC. The SRA software runs in its own Docker container in the target PC as well. SRA GUI is served as a single-page application through Ethernet port. The SRA container must have access to block devices and volumes to perform operations in the host file system.

Figure 6 illustrates the Docker Compose configuration file. Both system partitions and the boot partition are mounted into the container as devices. This allows SRA to write to current inactive system partition during the update process. An access to the boot partition is needed to update the boot loader configuration after successful write operation. The configuration directory in the host system is mounted to the container as volume. The container runs in privileged mode with elevated access to run system level operations inside the host filesystem.

```
sra.web.host:
  image: localhost:5000/sra
  restart: always
  privileged: true
  ports:
    - "44311:44311"
  depends_on:
    database:
      condition: service_healthy
  environment:
    App__ServerRootAddress: 'http://0.0.0.0:44311/'
  devices:
    - /dev/nvme1n1p1:/dev/nvme1n1p1 # boot
    - /dev/nvme1n1p2:/dev/nvme1n1p2 # system A
    - /dev/nvme1n1p3:/dev/nvme1n1p3 # system B
  volumes:
    - /boot/efi/config/:/config/ # configuration files
```

Figure 6. Docker compose configuration of SRA container

3.3 Configuring controller PC

To be able to utilize A/B software update procedure in the controller PC, a specific configuration is needed. This involves making an initial deployment installation when the PC is first set up with factory settings. The deployment is achieved by creating a bootable USB drive with any Linux operating system. In this project, Ubuntu is used due to its ability to run a minimized version of the

operating system without complete installation. The USB drive contains initial system image with LAC and SRA software and a Bash script to run a set of operations to make the initial deployment.

The deployment script (Appendix 1) initializes the controller PC by partitioning and formatting the hard drive according to the partition scheme (Figure 8). The deployment system image containing functioning operating system and all the Docker images, is installed to the system partition A. The script installs boot loader program and sets system A as bootable drive. A fstab file is created to define the mount points for the system A partition. Finally, a JSON file (Figure 7) is created to keep track of inactive and active partition of the hard drive. When the system is rebooted and USB drive is removed, the controller PC boots from system A partition and the SRA container is ready to perform a software update.

Figure 7. Configuration file

The partitioning is specified in the deployment script. It begins with a boot section, which contains the bootloader and other essentials for initiating the system's startup process. This is followed by two identical system partitions, which can contain fully functioning software versions. The last two section is reserved for persistent data to store user and application data accessible regardless of the active system partition.

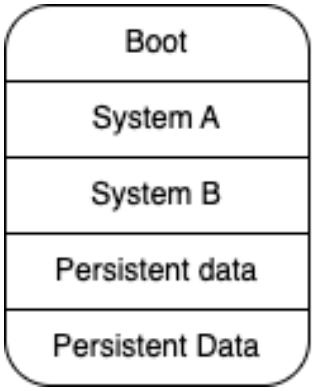

Figure 8. LAC PC partition scheme

3.4 Software update image

The controller update image is built in Azure DevOps pipeline. The system image contains functional Linux operating system and all the necessary software as Docker images to operate the controller PC. The software version of the image is written into a JSON file in this pipeline. The initial system image installed in deployment is built using the same pipeline.

3.5 SRA GUI

SRA GUI is a Vue.js single-page application that has components for uploading the software update image (Appendix 2) and an input field for SHA256 checksum. SHA256 checksum is used to verify successful upload process without data loss or corruption. The view has a component to display the current controller software version as well.

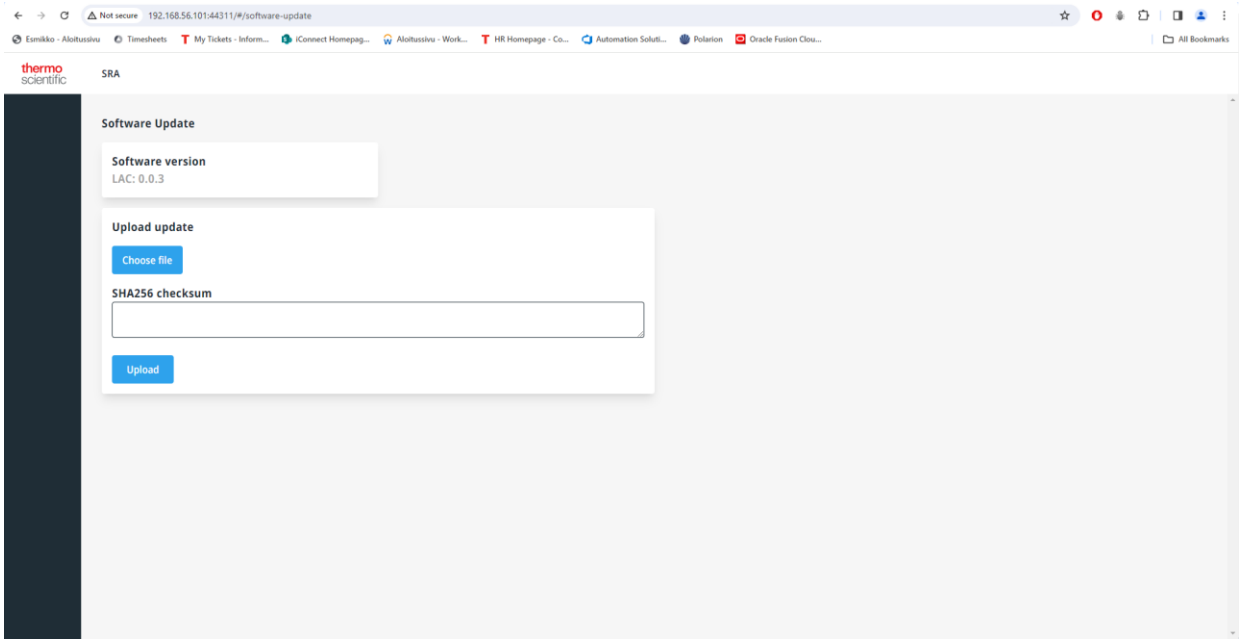

Figure 9. Interface for uploading software update

The disk image can be selected from local filesystem by clicking “Choose file” button. The actual generic HTML file input element is hidden, and clicking custom button element triggers the click event of the hidden input element. The selected file is validated for type and the file name is displayed in the view after confirming selection. The checksum can be typed or copy-pasted to the input field. When user clicks the upload button, the selected file and checksum is sent as form data to SRA backend. The status of the file upload is indicated in the upload component. If an error occurs during the upload process or the checksum verification Is unsuccessful, an error message is shown in the component.

3.6 SRA Backend

The SRA backend is built on the ASP.NET Boilerplate framework. The file uploaded by the user is handled by *SoftwareUpdateController* (Figure 10), a web controller, which utilizes *SoftwareUpdateService* through dependency injection for handling the update operations.

```
[Route( template: "api/[controller]")]
[ApiController]
1 reference
public class SoftwareUpdateController : SRAControllerBase
{
    private readonly ISoftwareUpdateService softwareUpdateService;

    0 references
    public SoftwareUpdateController(ISoftwareUpdateService softwareUpdateService)
    {
        this.softwareUpdateService = softwareUpdateService;
    }

    [HttpPost( template: "softwareUpdateFileUpload")]
    [RequestSizeLimit( bytes: long.MaxValue)]
    [RequestFormLimits(MultipartBodyLengthLimit = long.MaxValue)]
    0 references
    public async Task<ActionResult> SoftwareUpdateFileUploadAsync()
    {
        try
        {
            var multiPartBody :Stream = HttpContext.Request.Body;
            var contentType = MediaTypeHeaderValue.Parse( Request.ContentType);
            var multiPartBoundary :string = HeaderUtilities.RemoveQuotes(contentType.Boundary).Value;
            var update = new SoftwareUpdateImage() { Body = multiPartBody, Boundary = multiPartBoundary };

            var updateResult = await softwareUpdateService.UpdateSoftwareAsync(update);

            return Ok(updateResult);

        }
        catch (DirectoryNotFoundException ex)
        {
            return BadRequest(ex.Message);
        }
        catch (ValidationException ex)
        {
            return BadRequest(ex.Message);
        }
        catch (ArgumentException ex)
        {
            return BadRequest(ex.Message);
        }
    }
```

Figure 10. SoftwareUpdateController

SoftwareUpdateService (Figure 11) is responsible for writing the file into the inactive partition and checksum validation. The SRA application operates inside the active system partition that is fixed in size, in this configuration, 20 GB. The update image cannot be stored in memory or filesystem before the file is written to the inactive partition since update image is the same size as the active system partition. In ASP.NET framework this can be handled using streams. Streams handles files in chunks, as demonstrated in Code-Maze article (2024), which allows more effective memory management and enhance processing speed by enabling concurrent read and write operations on the file. However, a drawback of using this method is that it requires writing the image to the partition without validation by the backend software, as it would require writing it to the temporary storage.

```
public class SoftwareUpdateService : ISoftwareUpdateService
{
    private readonly IFileSystemManager fileSystemManager;


    0 references
    public SoftwareUpdateService(IFileSystemManager fileSystemManager)
    {
        this.fileSystemManager = fileSystemManager;
    }

    2 references
    public async Task<SoftwareUpdateResult> UpdateSoftwareAsync(SoftwareUpdateImage update)
    {
        var updateResult = await HandleMultipartSoftwareUpdateAsync(update);
        var newLacPcConfig = UpdateLacPcConfig(updateResult.UpdatedPartition);
        FileSystemManager.UpdateBootLoader(newLacPcConfig);
        updateResult.LacPcConfig = newLacPcConfig;

        return updateResult;
    }

    1 reference
    public async Task<SoftwareUpdateResult> HandleMultipartSoftwareUpdateAsync(SoftwareUpdateImage update)
    {
        var targetPartition = GetTargetPartition();

        var targetIsMounted bool = fileSystemManager.IsPartitionMounted(targetPartition.Path);
        if (targetIsMounted)
        {
            throw new ArgumentException( message: "target partition is mounted");
        }

        var multipartReader = new MultipartReader(update.Boundary, update.Body);
        var section = await multipartReader.ReadNextSectionAsync();

        var updateResult = new SoftwareUpdateResult();

        while (section != null)
        {
            if (ContentDispositionHeaderValue.TryParse( input: section.ContentDisposition, out var contentDisposition))
            {
                if (contentDisposition.IsFileDisposition())
                {
                    var fileSection = section.AsFileSection();
                    var extension string = Path.GetExtension(fileSection.FileName);
                    if (!AppConsts.AcceptedSoftwareImageExtensions.Contains(extension))
                    {
                        throw new ValidationException( message: "wrong file format");
                    }
                    updateResult.UpdatedPartition = await WriteFileToTargetPartitionAsync(fileSection.FileStream, targetPartition);

                }
                else
                {
                    updateResult.SourceChecksum = await new StreamReader(section.Body).ReadToEndAsync();
                }
            }

            section = await multipartReader.ReadNextSectionAsync();
        }

        updateResult.TargetChecksum = await CalculateSha256Checksum(updateResult.UpdatedPartition);
        if (updateResult.SourceChecksum != updateResult.TargetChecksum)
        {
            throw new ValidationException( message: "checksum failed");
        }

        return updateResult;
    }
```

Figure 11. SoftwareUpdateService

FileSystemManager (Figure 12), that is injected into the *SoftwareUpdateService*, provides helper functions for interaction with host file system. The manager can run Bash commands in the host operating system by utilizing .NET *Process* class. The partition configuration file, that was created when configuring LAC PC initially, is read to decide which partition the file is written. The inactive partition path, for example /dev/sda2, is checked if it is mounted to the filesystem, to avoid writing to the active partition. After successful writing, the partition configuration file is updated, and the new configuration is sent to the user. A successful writing operation means that no exceptions are thrown during the writing process and the checksum of the updated partition, and the user provided SHA256 checksum matches. In the end of the process, SRA runs a script in the host filesystem using *chroot* to update the settings to be able to boot from the updated partition on the next reboot.

```
public class FileSystemManager : IFileSystemManager
{

    3 references
    public static string RunShellCommand(string commandName, string arguments)
    {
        using (var process = new Process())
        {
            process.StartInfo.FileName = commandName;
            process.StartInfo.Arguments = arguments;
            process.StartInfo.UseShellExecute = false;
            process.StartInfo.RedirectStandardOutput = true;
            process.StartInfo.CreateNoWindow = true;
            process.Start();

            var output :string = process.StandardOutput.ReadToEnd();
            process.WaitForExit();
            return output;
        }
    }

    4 references
    public LacPcConfig ReadLacPcConfig(string path)
    {
        if (string.IsNullOrEmpty(path))
        {
            throw new ArgumentException( message: "Path for LAC PC configuration file is not defined");
        }
        var jsonString = File.ReadAllText(path);
        var lacPcConfig = JsonConvert.DeserializeObject<LacPcConfig>(jsonString);
        return lacPcConfig;
    }

    2 references
    public void WriteLacPcConfig(LacPcConfig lacPcConfig, string configPath)
    {
        var jsonString = JsonConvert.SerializeObject(lacPcConfig);
        File.WriteAllText(configPath,  contents: jsonString);
    }

    2 references
    public string GetDeviceUuidByPath(string path)
    {
        if (string.IsNullOrEmpty(path)) { return null; }
        var arguments = $"{path} -s UUID -o value";
        var uuid :string  = RunShellCommand("blkid", arguments);
        return uuid.Trim();
    }

    2 references
    public bool IsPartitionMounted(string path)
    {
        var output :string  = RunShellCommand("mount",  arguments: "");
        return (output.Contains(path));

    }

    1 reference
    public static void UpdateBootLoader(LacPcConfig lacPcConfig)
    {
        var newActive :DiskPartition  = lacPcConfig.LacPcPartitions["Active"];
        var boot :DiskPartition  = lacPcConfig.LacPcPartitions["Boot"];
        RunShellCommand("chroot",
             arguments: $"/host bash                /update_lac_settings.sh {boot.Path} {newActive.Path}");
    }

}
```

Figure 12. FileSystemManager

The script for post-update configuration (Figure 14) requires two arguments: the path to the boot partition and the path to the updated partition. The script mounts the new partition, that now contains the uploaded update image and runs a set of tasks in the new operating system. Firstly, it reads the UUIDs of each partition and defines the mount points for the new partition by creating a fstab file. Secondly, it copies the old network interface settings from the previous active partition to ensure communication after reboot. Finally, the script runs a sequence of commands that updates the GRUB bootloader to enable booting from the newly updated partition. In this sequence,

the boot partition is mounted to the filesystem of the new partition. Essential system directories, from currently active filesystem are bind-mounted. These directories, for example, contain communication with the kernel to ensure the that the new system partition can run programs during the settings update. In the end of the sequence *chroot* command is used to run *grub-install* in the new filesystem. When the system is rebooted, controller PC boots in system partition B and the new updated SRA container is ready to perform another software update.

```
#!/bin/bash
BOOT_PART=$1
NEW_PART=$2

echo "updating settings for ${NEW_PART}"

sudo mount $NEW_PART /mnt

UUID_P1=$(sudo blkid -s UUID -o value ${BOOT_PART})
UUID_PNEW=$(sudo blkid -s UUID -o value ${NEW_PART})


echo "creating fstab"
sudo cp /dev/null /mnt/etc/fstab
echo "UUID=${UUID_P1}   /boot/efi      vfat       defaults      0          1" | sudo tee -a /mnt/etc/fstab
echo "UUID=${UUID_PNEW} /              ext4       defaults      0          1" | sudo tee -a /mnt/etc/fstab

echo "copying network settings"
sudo cp /etc/network/interfaces /mnt/etc/network/interfaces

echo "updating bootloader"
sudo mkdir /mnt/boot/efi
sudo mount ${BOOT_PART} /mnt/boot/efi
sudo mount --bind /dev /mnt/dev
sudo mount --bind /dev/pts /mnt/dev/pts
sudo mount --bind /proc /mnt/proc
sudo mount --bind /sys /mnt/sys
sudo chroot /mnt grub-install --target=x86_64-efi --efi-directory=/boot/efi
sudo chroot /mnt update-grub
```

Figure 13. Script for post-update configuration

4 Results and Conclusion

4.1 Results

The SRA development was launched and has a framework for future expansion. A straightforward GUI was implemented where user can upload the software update image. The image was streamed and written in place to the inactive partition, without the need for temporary storage. Ensuring the integrity of the file transfer, a checksum verification was implemented. Finally, the

bootloader was successfully, updated and system was able to boot successfully from the newly updated system image. The update process was completed seamlessly, as the only downtime for the controller PC was during the rebooting phase.

Implementation of the software update feature in this project provided one approach to such updates, though not necessarily the optimal one. While the system functioned adequately, there is room for refinement. The implementation followed a "happy path" scenario, achieving successful file transfer and boot process without taking into consideration the potential error cases. These cases could include boot failures, and unhealthy operating system or higher-level software.

In a more refined model, the update process could be more structured and rely more on operating system-level functionality rather than custom scripts. Allowing Docker container, that is designed to be isolated environment, to execute code and run scripts in host operating system in privileged mode could be a cybersecurity risk. Such interactions can potentially expose the host to unauthorized access.

4.2 Conclusion

The development process was carried as part of a cross-functional Scrum team. Although the work was carried out individually, support and feedback by other team members were provided if necessary. Throughout the development cycle, the progress was regularly presented to management and product owners as demo presentations.

In the beginning of this project, final decision regarding the deployment of the controller PC has not been made, which brought some challenges. However, these challenges brought valuable information to the company, as this thesis included exploration on the deployment process of LAC PC and configuring support for A/B updates.

Implementing the application containing the first feature proved to be a valuable learning experience and was an effective method for initial exploration of seamless software updates. The knowledge of Linux fundamentals turned out to be essential in this project and allowed to deep dive into Linux systems. This will be beneficial in future SRA development as the experience gained

through this project can be carried forward in future features that might require interaction with the operating system.

Next steps in refining the update process could include implementing an automatic rollback mechanism in case of boot failure. This feature came up in multiple sources in research phase of seamless software updates. Naturally, it is an essential to secure the reliability of the laboratory automation system. Additionally, the process should be scalable, as eventually SRA should be able to initiate the software update process to the whole automation system.

4.3 Reliability and ethics

Using artificial intelligence tools can improve learning and students are encouraged to use AI to develop their working life skills (Arene, 2023). During the writing of this thesis, ChatGPT was used to provide feedback. In the beginning of the writing process, it was used to plan the thesis structure. Additionally, it was used to refine text and vocabulary by suggesting synonyms and restructuring sentences.

When using AI tools, the practices of scientific writing must be followed. Ideas, information, words, or other material of others cannot be published without a reference. Artificial intelligence cannot be used as a source for scientific text, and authors are responsible for the accuracy of their works. (Arene, 2023) These ethical guidelines were taken into consideration in this thesis process. AI tools were not used to retrieve information in this thesis. When refining text, the original source was checked to not alter the original writer's information.

References

.NET Foundation. (n.d). What is the ASP.NET Boilerplate? .NET Foundation. Retrieved April 20, 2024 from https://aspnetboilerplate.com/Pages/Documents

Android. (2024). A/B (seamless) system updates. Android. Retrieved April 20, 2024 from https://source.android.com/docs/core/ota/ab

Arene, (2023). Arene's recommendations on the use of artificial intelligence for universities of applied sciences. Arene. Retrieved June 3, from https://arene.fi/wp-content/uploads/PDF/2023/AI-Arene-suositukset_EN.pdf?_t=1686641192

Banik, S., & Zimmer, V. (2022). System Firmware: An Essential Guide to Open Source and Embedded Solutions. Apress.

Breshnan, C., & Blum, R. (2021). Mastering Linux System Administration. John Wiley & Sons, Incorporated.

Byrne, D. (2018). Updating Linux using A/B Partitions. David Byrne's Blog. Retrieved April 20, from https://blog.davidbyrne.dev/2018/08/16/linux-ab-partitions

Code Maze. (2024). Uploading Large Files in ASP.NET Core. Code-Maze. Retrieved May 25, from https://code-maze.com/aspnetcore-upload-large-files/

DigitalOcean. (2024). Top 50+ Linux Commands You MUST Know. Retrieved May 29, from https://www.digitalocean.com/community/tutorials/linux-commands

Docker. (n.d). Docker overview. Docker Inc. Retrieved April 23, 2024 from **https://docs.docker.com/get-started/overview/**

Flewelling, P. (2018). The Agile Developer's Handbook : Get More Value from Your Software Development: Get the Best Out of the Agile Methodology. Packt Publishing, Limited.

Javapoint. (n.d). Linux/Unix: chroot Command. Retrieved May 29, from https://www.javatpoint.com/linux-chroot-command

Jena, B. K. (2023). A Definitive Guide to Learn The SHA-256 (Secure Hash Algorithms). Retrieved May 29, from https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm

McGrath, M. (2019). Bash in Easy Steps : Utilize the Power of Bash. In Easy Steps.

Pasternak, T. (2024). A/B updates. Medium. Retrieved from https://medium.com/@pasternak-tal/a-b-updates-254e891d3d0b

Red Hat (n.d). Bind Mounts and Context-Dependent Path Names. Retrieved May 29, from https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/global_file_system_2/s1-manage-pathnames

Ryder, T. (2018). Bash Quick Start Guide : Get up and Running with Shell Scripting with Bash. Packt Publishing, Limited.

Smith, R. W. (2012). Linux Essentials. John Wiley & Sons, Incorporated.

Technology Networks. (2017). Key Challenges and Pain Points in the Global Laboratory Market. Technology Networks. Retrieved May 29, from https://www.technologynetworks.com/analysis/articles/key-challenges-and-pain-points-in-the-global-laboratory-market-291108

Vue.js. (n.d). What is Vue? Vue.js. Retrieved April 20, 2024. https://vuejs.org/guide/introduction.html.

Ward, B. (2021). How Linux Works: What Every Superuser Should Know (3rd Edition). No Starch Press.

Yogesh, B. (2020). Hands-On Booting: Learn the Boot Process of Linux, Windows, and Unix. Apress.

Appendices

Appendix 1. Script to configure LAC PC

```
#!/bin/bash

# Update and install necessary tools
sudo apt-get update
sudo apt-get install -y qemu-utils parted dosfstools grub-efi-amd64-bin

# Associate the image with a particular hard drive
HARD_DRIVE=/dev/nvme1n1
echo "Chosen hard drive: $HARD_DRIVE"

echo "Create the partition table"
sudo parted $HARD_DRIVE mklabel gpt

echo "Create the partitions"
sudo parted $HARD_DRIVE mkpart primary fat32 0% 1GiB
sudo parted $HARD_DRIVE mkpart primary ext4 1GiB 21GiB
sudo parted $HARD_DRIVE mkpart primary ext4 21GiB 41GiB
sudo parted $HARD_DRIVE mkpart primary ext4 41GiB 191GiB
sudo parted $HARD_DRIVE mkpart primary ext4 191GiB 238GiB

echo "Format the partitions"
sudo mkfs.vfat ${HARD_DRIVE}p1
sudo mkfs.ext4 ${HARD_DRIVE}p2
sudo mkfs.ext4 ${HARD_DRIVE}p3
sudo mkfs.ext4 ${HARD_DRIVE}p4
sudo mkfs.ext4 ${HARD_DRIVE}p5

echo "Mount and install the LAC partition image"
sudo dd if=lac_active_partition.img of=${HARD_DRIVE}p2 bs=4M status=progress

echo "Set boot flag"
sudo parted ${HARD_DRIVE} print
sudo parted ${HARD_DRIVE} set 1 boot on
sudo parted ${HARD_DRIVE} print

echo "Install GRUB"
sudo mount ${HARD_DRIVE}p2 /mnt
sudo mkdir -p /mnt/boot/efi/
sudo mount ${HARD_DRIVE}p1 /mnt/boot/efi
sudo mount --bind /dev /mnt/dev
sudo mount --bind /dev/pts /mnt/dev/pts
sudo mount --bind /proc /mnt/proc
sudo mount --bind /sys /mnt/sys
sudo chroot /mnt /usr/bin/apt-get update -y
sudo chroot /mnt /usr/bin/apt-get install -y grub-efi-amd64
sudo chroot /mnt/ grub-install --target=x86_64-efi --efi-directory=/boot/efi --bootloader-id=GRUB
sudo chroot /mnt/ grub-mkconfig -o /boot/grub/grub.cfg
sudo umount /mnt/dev/pts
sudo umount /mnt/dev
sudo umount /mnt/proc
sudo umount /mnt/sys


echo "Define the mount points"
sudo cp /dev/null /mnt/etc/fstab
UUID_P1=$(sudo blkid -s UUID -o value ${HARD_DRIVE}p1)
UUID_P2=$(sudo blkid -s UUID -o value ${HARD_DRIVE}p2)
echo "UUID=${UUID_P1}  /boot/efi       vfat    defaults        0       1" | sudo tee -a /mnt/etc/fstab
echo "UUID=${UUID_P2}  /               ext4    defaults        0       1" | sudo tee -a /mnt/etc/fstab
sudo cat /mnt/etc/fstab

echo "Create LAC PC config file"
sudo mkdir -p /mnt/boot/efi/config
sudo echo '{ "LacPcPartitions": {
    "Boot": { "Path":  "'"${HARD_DRIVE}p1"'", "Uuid": "'"${UUID_P1}"'" },
    "Active": { "Path": "'"${HARD_DRIVE}p2"'", "Uuid": "'"${UUID_P2}"'" },
    "Inactive": { "Path": "'"${HARD_DRIVE}p3"'", "Uuid": "" },
    "Data": { "Path": "'"${HARD_DRIVE}p4"'", "Uuid": "" },
    "Logs": { "Path": "'"${HARD_DRIVE}p5"'", "Uuid": "" } } }' > /mnt/boot/efi/config/lac_pc_config.json
sudo cat /mnt/boot/efi/config/lac_pc_config.json

sudo umount /mnt/boot/efi
sudo umount /mnt
```

Appendix 2. Vue.js component for uploading file

```
<script lang="ts">
export default defineComponent({
    },
    methods: {
        onBrowseButtonClick() {
            (this.$refs.fileInput as HTMLInputElement).click()
        },
        onFileInput(event: Event) {
            this.status = UploadStatus.Idle
            const target = (event.target as HTMLInputElement)
            const inputFile =  (target.files as FileList)[0]
            if (inputFile && isValidFileExtension(inputFile.name, this.acceptedFileTypes)){
                this.file = inputFile
            }
            else{
                this.file = null
                this.status = UploadStatus.Error
                this.statusMessage = "invalid input"
            }
        },
        async onUploadButtonClick () {
            if (this.file) {
                this.status = UploadStatus.Uploading
                this.statusMessage = "uploading"
                const formData = new FormData()
                formData.append("SoftwareUpdateFile", this.file)
                formData.append("Checksum", this.checksum)
                try {
                    await ajax.post("/api/SoftwareUpdate/softwareUpdateFileUpload",
                        formData, {
                            headers:
                                {'Content-Type': 'multipart/form-data'}
                            })
                    this.status = UploadStatus.Success
                    this.statusMessage = "success"
                    this.file = null
                }
                catch (error: any){
                    this.status = UploadStatus.Error
                    if (error.response && error.response.data){
                        this.statusMessage = error.response.data.result
                    }
                    else if (axios.isAxiosError(error)){
                        this.statusMessage = error.message
                    }
                    else{
                        this.statusMessage = "Something went wrong"
                    }
                    console.error(error)
                }
            }
            else {
                this.status = UploadStatus.Error
                this.statusMessage = "No file selected"
```