University POLITEHNICA of Bucharest

Faculty of Automatic Control and Computers,
Computer Science and Engineering Department

Diploma

Minimizing Dockerfiles

Scientific Adviser:
Prof. Răzvan Deaconescu

Author:
Andrei Petrea

Bucharest, 2025

I want first and foremost to thank my advisor, Prof. Răzvan Deaconescu, for his guidance and support throughout this project. His expertise and insights have been invaluable in shaping my understanding of the subject matter and in helping me navigate the challenges I faced during this endeavor.

I also want to thank my friends and family without whom these 4 years would have been so much harder.

Contents

List of Figures

List of Listings

Chapter 1

Introduction

Chapter 2

Background

2.1 A brief history of containers

2.2 The anatomy of a container

```
1 FROM python:3.10.0-slim
2 COPY requirements.txt .
3 RUN pip install -r requirements.txt
4 COPY . .
5 CMD ["python", "adapter.py"]
```

Listing 2.1: Sample Dockerfile

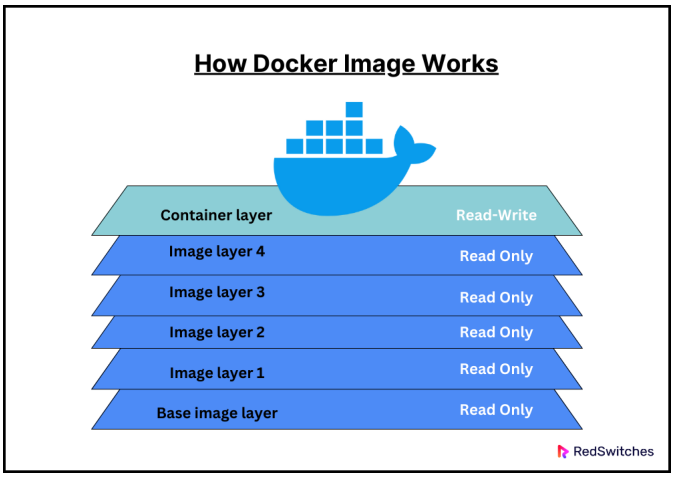

Figure 2.1: Container layers

```
1     FROM alpine:latest AS builder
2     RUN apk --no-cache add build-base
3
4     FROM builder AS build1
5     COPY source1.cpp source.cpp
6     RUN g++ -o /binary source.cpp
7
```

```
8     FROM builder AS build2
9     COPY source2.cpp source.cpp
10    RUN g++ -o /binary source.cpp
```

Listing 2.2: Sample Multi-stage Dockerfile

2.3 Related work

Chapter 3

Motivation and Objectives

Chapter 4

Use Cases

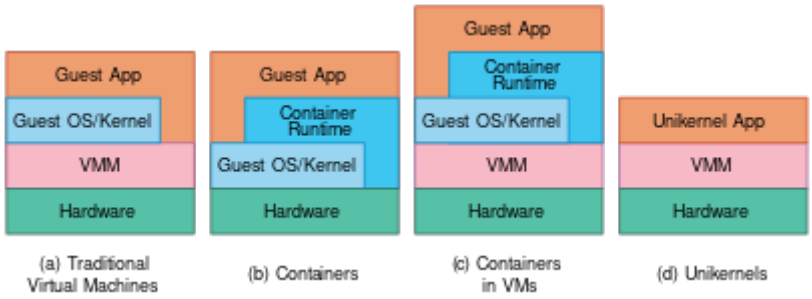

Figure 4.1: High-level comparison of the software components of traditional VMs (a), containers (b), containers in VMs (c) with unikernels (d).

```
1    spec: v0.6
2    runtime: base:latest
3    rootfs: ./Dockerfile
4    cmd: ["/helloworld"]
```

Listing 4.1: Sample Kraftfile

Chapter 5

Building Blocks

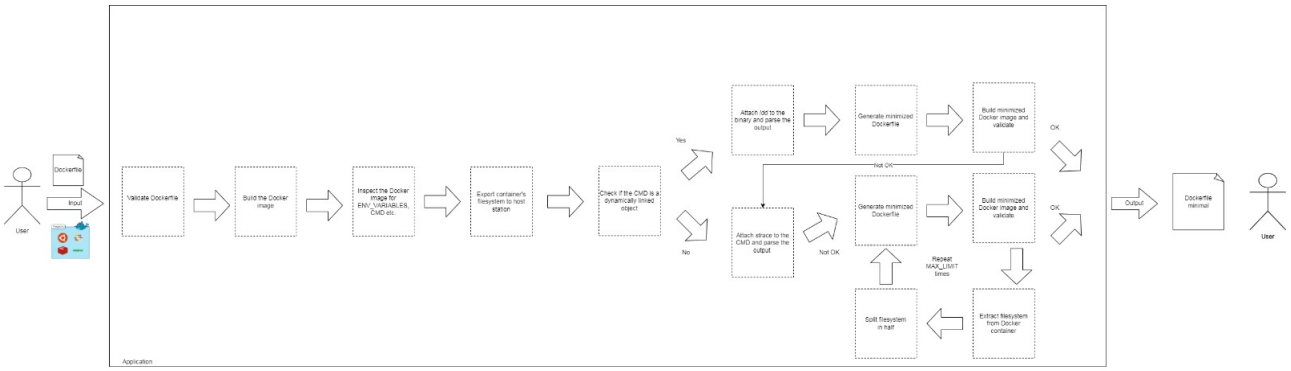

Figure 5.1: High-level architecture of the application

Chapter 6

Architecture Overview

6.1 Static Analysis

```
~ > ldd /usr/bin/man
    linux-vdso.so.1 (0x00007ffe831f5000)
    libmandb-2.9.1.so => /usr/lib/man-db/libmandb-2.9.1.so (0
        x00007f068b572000)
    libman-2.9.1.so => /usr/lib/man-db/libman-2.9.1.so (0
        x00007f068b52f000)
    libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007f068b502000)
    libpipeline.so.1 => /lib/x86_64-linux-gnu/libpipeline.so.1 (0
        x00007f068b4f1000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f068b2ff000)
    libgdbm.so.6 => /lib/x86_64-linux-gnu/libgdbm.so.6 (0
        x00007f068b2ef000)
    libseccomp.so.2 => /lib/x86_64-linux-gnu/libseccomp.so.2 (0
        x00007f068b2cb000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f068b59b000)
```

Listing 6.1: ldd command

6.2 Dynamic Analysis

```
~ > strace -fe execve,openat echo "Hello, World\!"
execve("/usr/bin/echo", ["echo", "Hello, World!"], 0x7fff43b04ce8 /* 36
    vars */) = 0
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC)
    = 3
Hello, World!
+++ exited with 0 +++
```

Listing 6.2: strace command

6.3 Brute Force

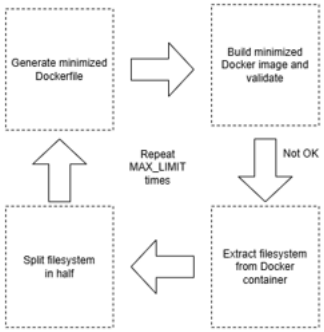

Figure 6.1: Brute force approach to finding the minimal Dockerfile

Chapter 7

Implementation Details

7.1 Overview

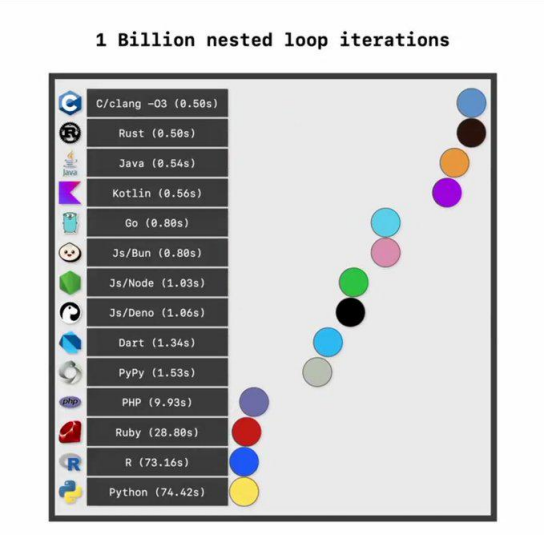

Figure 7.1: Language execution speed comparison [1]

7.2 Project Structure

```
.
|-- cmd
  |-- main.go
|-- dockerminimizer
|-- dockerminimizer.go
|-- go.mod
|-- go.sum
|-- install.sh
|-- ldd
  |-- ldd.go
|-- logger
  |-- logger.go
|-- preprocess
```

```
      |-- preprocess.go
    |-- README.md
    |-- strace
      |-- strace.go
    |-- types
      |-- types.go
    |-- utils
      |-- utils.go
```

Listing 7.1: Project Structure

7.3 Preprocessing

```
    type DockerConfig struct {
        User         string                    `json:"User"`
        ExposedPorts map[string]map[string]any `json:"ExposedPorts"`
        Env          []string                  `json:"Env"`
        Cmd          []string                  `json:"Cmd"`
        WorkingDir   string                    `json:"WorkingDir"`
        Entrypoint   []string                  `json:"Entrypoint"`
}
```

Listing 7.2: Go DockerConfig structure

```
FROM node:20.9.0 as builder

WORKDIR /app

COPY app.js /app/app.js
COPY package.json /app/package.json
COPY package-lock.json /app/package-lock.json

RUN npm install
EXPOSE 3000
CMD ["node", "app.js"]


FROM scratch

ENV PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
ENV NODE_VERSION=20.9.0
ENV YARN_VERSION=1.22.19
WORKDIR /app
EXPOSE 3000/tcp
ENTRYPOINT ["docker-entrypoint.sh"]
CMD ["node", "app.js"]
```

Listing 7.3: Dockerfile.minimal.template

7.4 Static Analysis

```
    func ParseOutput(output []byte, rootfsPath string) (map[string][]string,
         map[string]string) {
        files := make(map[string][]string)
        symLinks := make(map[string]string)
        scanner := bufio.NewScanner(bytes.NewReader(output))
```

```
5           for scanner.Scan() {
6               line := scanner.Text()
7               if strings.Contains(line, "=>") {
8                   parts := strings.Split(line, "=>")
9                   lib := strings.Split(strings.TrimSpace(parts[1]), " ")[0]
10
11                  if strings.Contains(lib, "not found") {
12                      continue
13                  }
14                  utils.AddFilesToDockerfile(lib, files, symLinks, rootfsPath)
15
16              } else if strings.Contains(line, "not found") {
17                  continue
18              } else {
19                  lib := strings.Split(strings.TrimSpace(line), " ")[0]
20                  utils.AddFilesToDockerfile(lib, files, symLinks, rootfsPath)
21              }
22          }
23          return files, symLinks
24      }
```

Listing 7.4: ldd parser function

```
1   func RealPath(path string) string {
2           realPath, _ := filepath.Abs(path)
3           return filepath.Clean(realPath)
4   }
5
6   func CheckIfFileExists(file string, envPath string) bool {
7           info, err := os.Stat(envPath + "/" + file)
8           return !os.IsNotExist(err) && !info.IsDir()
9   }
10
11  func CheckIfSymbolicLink(file string, envPath string) bool {
12          info, err := os.Lstat(envPath + "/" + file)
13          if err != nil {
14                  return false
15          }
16          return info.Mode()&os.ModeSymlink != 0
17  }
18
19  func ReadSymbolicLink(file string, envPath string) string {
20          link, _ := os.Readlink(envPath + "/" + file)
21          resolved := link
22          if !filepath.IsAbs(link) {
23                  resolved = filepath.Join(filepath.Dir(envPath+"/"+file),
                        link)
24          }
25          return strings.TrimPrefix(resolved, envPath)
26  }
27
28  func AddFilesToDockerfile(file string, files map[string][]string, symLinks
        map[string]string, rootfsPath string) {
29          file = RealPath(file)
30          if CheckIfFileExists(file, rootfsPath) {
31                  if CheckIfSymbolicLink(file, rootfsPath) {
32                          symLinks[file] = ReadSymbolicLink(file, rootfsPath)
33                  } else {
```

```
34                          files[filepath.Dir(file)] = AppendIfMissing(files[
                                filepath.Dir(file)], file)
35                  }
36          }
37  }
```

Listing 7.5: File util functions

7.5 Dynamic Analysis

For this stage, we attach a *strace* probe to the docker running command, in order to determine the files used and processes that are spawned by the command. In order to do this, we utilize Docker's volumes to mount a statically-linked *strace* binary into the container, as well as a logfile to save the output of the command. The reason for using a statically-linked binary is so that the libraries used by strace don't interfere with the libraries used by the command. To determine the runtime dependencies of the command, we will trace the *open* and *exec* familly of syscalls [2] as well as passing the *-f* flag to trace the child processes as well.

```
1     func parseOutput(output string, syscalls []string, files map[string][]
          string, symLinks map[string]string, envPath string) {
2         regexes := make(map[string]*regexp.Regexp)
3         for _, syscall := range syscalls {
4             regexes[syscall] = regexp.MustCompile(syscall + `
                  \([^"]*?"([^"]+)"`)
5         }
6         for line := range strings.SplitSeq(output, "\n") {
7             for _, syscall := range syscalls {
8                 if regexes[syscall].MatchString(line) {
9                     match := regexes[syscall].FindStringSubmatch(line)
10                    if len(match) > 1 {
11                        utils.AddFilesToDockerfile(match[1], files, symLinks
                              , envPath+"/rootfs")
12                    }
13                    break
14                }
15            }
16        }
17    }
```

Listing 7.6: File util functions

```
1     func getSheBang(command string, rootfsPath string) string {
2         file, err := os.Open(rootfsPath + "/" + command)
3         if err != nil {
4                 log.Error("Failed to open file:", command)
5                 return ""
6         }
7         defer file.Close()
8         var firstLine []byte
9         buf := make([]byte, 1)
10        for {
11                n, err := file.Read(buf)
12                if n > 0 {
13                        if buf[0] == '\n' {
14                                break
15                        }
16                        firstLine = append(firstLine, buf[0])
```

```
17                  }
18                  if err != nil {
19                          break
20                  }
21          }
22          return string(firstLine)
23  }
24
25  func parseShebang(imageName string, containerName string, syscalls []string,
26          files map[string][]string, symLinks map[string]string, envPath
                string, metadata types.DockerConfig, timeout int) (map[string][]
                string, map[string]string) {
27          command := utils.GetContainerCommand(imageName, envPath, metadata)
28          hasSudo := utils.HasSudo()
29          shebang := getSheBang(command, envPath+"/rootfs")
30          regex := regexp.MustCompile(`^#!\s*([^\s]+)`)
31          if !regex.MatchString(shebang) {
32                  log.Error("Failed to find shebang in file:", command)
33                  return files, symLinks
34          }
35          match := regex.FindStringSubmatch(shebang)
36          if len(match) < 2 {
37                  log.Error("Failed to find interpreter in shebang:", command)
38                  return files, symLinks
39          }
40          interpreter := match[1]
41          lddCommand := hasSudo + " chroot " + envPath + "/rootfs ldd " +
                interpreter
42          log.Info("Running command:", lddCommand)
43          lddOutput, err := exec.Command("sh", "-c", lddCommand).
                CombinedOutput()
44          if err != nil {
45                  log.Error("Failed to run ldd command\n" + err.Error())
46          }
47          files, symLinks = ldd.ParseOutput(lddOutput, envPath+"/rootfs")
48
49          output := getStraceOutput(imageName, envPath+"/strace", envPath+"/
                log.txt", syscalls,
50                  containerName, interpreter, envPath, metadata, timeout)
51          parseOutput(output, syscalls, files, symLinks, envPath)
52          return files, symLinks
53  }
```

Listing 7.7: File util functions

7.6 Binary Search

```
1   func binarySearchStep(envPath string, context string, timeout int, step int,
        usedFiles map[string][]string,
2           unusedFiles map[string][]string) (map[string][]string, map[string][]
                string, error) {
3           for {
4                   if len(unusedFiles) == 0 {
5                           return nil, nil, errors.New("no unused files or
                                symbolic links left to process")
6                   }
```

```
7                 usedFiles, unusedFiles = splitFilesystem(usedFiles,
                      unusedFiles)
8                 filename := fmt.Sprintf("Dockerfile.minimal.binary_search.%d
                      ", step)
9                 usedFiles = utils.ShrinkDictionary(usedFiles)
10                utils.CreateDockerfile(filename, "Dockerfile.minimal.
                      template", envPath, usedFiles, nil)
11                err := utils.ValidateDockerfile(filename, envPath, context,
                      timeout)
12                if err == nil {
13                        return make(map[string][]string), usedFiles, nil
14                }
15        }
16 }
17
18 func BinarySearch(envPath string, maxLimit int, context string, timeout int)
       error {
19        log.Info("Starting binary search...")
20        usedFiles, unusedFiles, err := parseFilesystem(envPath + "/rootfs")
21        if err != nil {
22                log.Error("Error parsing filesystem:", err)
23                return errors.New("error parsing filesystem")
24        }
25        step := 1
26        for i := range maxLimit {
27                log.Info("Binary search iteration:", step)
28                usedFiles, unusedFiles, err = binarySearchStep(envPath,
                      context, timeout, step,
29                        usedFiles, unusedFiles)
30                if err != nil {
31                        break
32                }
33                step = i + 1
34        }
35        if step == maxLimit {
36                log.Info("Reached maximum limit of binary search iterations:
                      ", maxLimit)
37                return errors.New("reached maximum limit of binary search
                      iterations")
38        }
39        log.Info("Binary search completed sucessfully.")
40        return nil
41 }
```

Listing 7.8: Binary search

Chapter 8

Performance Evaluation

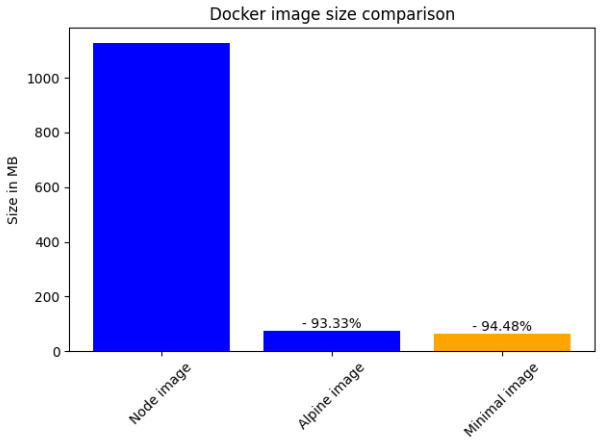

Figure 8.1: Minimal dockerfile comparision

Chapter 9

Status and Planned Work

9.1 Status

9.2 Planned Work

Bibliography

[1] Ben Dicken. Speed comparison of programming languages. https://benjdd.com/languages/.

[2] Adam Rehn. Identifying application runtime dependencies. https://unrealcontainers.com/blog/identifying-application-runtime-dependencies.