Virtual Development Environment in a Box
Software Requirements Specification

Ian Orzel
Dylan McDougall

October 2022

Contents

Chapter 1

Introduction

1.1 Purpose

The purpose of this document is to specify the requirements that we believe the developed software should satisfy. This document serves as a contract between developer and client to agree on what this software should look like. This document should be used by the developers and client during the development process in order to ensure that the software satisfies all of the requirements listed in this document.

1.2 Scope

This document describes the required specifications for the Virtual Development in a Box project. The software produced by this project will allow users to:

- Download and run containers containing preloaded environments tailored to the end user's classes on their local machine.
- Use development tools, such as gcc, make, gdb, gld, and gas, within the container for use in program development and testing.
- Run containers which emulate a system architecture that is different from the native architecture of the end user's local machine.
- Run your containers on Windows, MacOS, and Debian-based Linux.
- Import and export files to and from the container as well as provide standard input and output. For instance, in a compiler class, be able to add your compiler and program to the container to be compiled and interactively run it using standard input and output.
- Use pre-built containers for the Compiler Theory, Operating System Concepts, and Programming Language Concepts classes. Also, create con-

tainers for new courses or for existing courses without having to create an entire virtual environment from scratch and share them with students.

This application will be used by students and faculty at Florida Tech in order to facilitate students possessing development environments to be used for completing work in their classes. Certain classes require particular tools and environments for students to complete coursework. It can often be difficult for both student and faculty to set up these environments for the courses. Thus, this project hopes to address this by simplifying the process for students to setup and use these development environments. This is done by creating containers that contain all of the needed resources for students to complete their work.

1.3 Definitions, Acronyms, and Abbreviations

- "The application" or "the program" refers to the Python program that the end user interacts with directly, which facilitates interactions between the end user and the containers.
- The "system" refers to the general set of software components required for the application to run as well as the application itself.
- A "container" or "image" is a preconfigured QEMU virtual machine, stored as a virtual hard disk image and a configuration file on the end user's local machine.
- To "run" a container means to execute the appropriate QEMU system with the necessary flags stored in the container's configuration file such that it boots from the container's virtual hard disk image.
- The "local machine" is the computer/operating system which the application is directly being executed by.

1.4 Overview

This document provides a description of the project, its uses, its constraints, our assumptions, and a detailed list of requirements functional, performance, and logical requirements.

Chapter 2

Description

2.1 Product Perspective

This product will be self-contained and not tied to any pre-existing system. We will be making this new system from scratch using known tools.

Users will be able to interface into this system using a provided command line tool. This will allow users to utilize a series of commands in order to take advantage of the functionality of the system. This interface will be available to users on Windows, MacOS, and Debian-based Linux operating systems.

2.2 User Characteristics

The users of this product will be students and faculty at Florida Tech. The typical user will be pursuing an undergraduate degree in computer science, but it is assumed that they have limited technical skill, as some may be first year students. It will also be assumed that the user base has limited experience using command-line tools.

2.3 Constraints

The portability of the application is limited to those platforms for which Python and Qemu are available.

The application is expected to be able to run and interact with containers locally on an average laptop or desktop computer typical of a university student. The application must be able to run on a system with 8GB+ of RAM and a relatively recent Intel, AMD, or Apple CPU with at least 4 cores. When running a container, it must not consume so many resources that the end user is unable to comfortably multitask. In the case of a laptop, running a container should not consume overly excessive amounts of power to the point where using

the program while disconnected from wall power becomes impractical. Storing containers should also not consume an excessive amount of the end user's permanent storage.

2.4 Assumptions and Dependencies

It is assumed that the end user is using one of the following as the operating system installed on their local machine:

- Windows 10 or newer
- macOS Monterey or newer
- Ubuntu 20.04 LTS or newer (or equivalent binary compatible Ubuntu distribution)
- Debian 11 or newer (or equivalent binary compatible Debian distribution)

During the installation of the application and when downloading containers, it is assumed that the end user is connected to a non-metered internet connection. (An internet connection is, however, not required for running the containers once they are downloaded onto the local machine.)

It is assumed that, at all times, the system has read and write access to the current user's home/user directory.

Aside from the operating systems, the application is dependent on the following software:

- QEMU 7 or newer
- Python 3.7 or newer
- 7-zip 16 or newer
- Bash (macOS/Linux only)
- cURL (macOS/Linux only)

Chapter 3

Requirements

3.1 Requirements for Specific Containers

Here, we will describe the requirements related to the containers that will be delivered along with our system. These containers are specially designed to accommodate particular classes at Florida Tech. We will describe the requirements for containers for the classes of Compiler Theory, Operating Systems Concepts, and Programming Language Concepts.

3.1.1 Compiler Theory

Req. 1 (Functional)**.** *The container must contain the GNU Toolchain and the GNU Binutils and be accessible to be run by the user.*

Req. 2 (Functional)**.** *The container must have access to the "malloc" Linux function, which will allow the owner of the container to arbitrarily allocate memory in the container.*

Req. 3 (Functional)**.** *The container must have the ability to compile and run Java programs from within it.*

Req. 4 (Functional)**.** *The container must be able to run binaries that are built for the SPARC architecture.*

3.1.2 Operating Systems Concepts

To be determined after further discussion with Dr. Ribiero. We have not had the opportunity to discuss this with him to determine the needed requirements. We plan to hold this discussion until the second semester and then will modify this section.

3.1.3 Programming Language Concepts

Req. 5 (Functional). *The container must be able to compile and run programs written in the Go programming language.*

Req. 6 (Functional). *The container must be able to compile and run programs written in the Rust programming language.*

Req. 7 (Functional). *The container must be able to compile and run programs written in the Ada programming language.*

Req. 8 (Functional). *The container must be able to compile and run programs written in the Haskell programming language.*

3.2 Container Management

Req. 9 (Functional). *Users must be able to download any container available on the server onto their local machine for use.*

Req. 10 (Functional). *Users must be able to start an arbitrary container that they have downloaded in order to use.*

Req. 11 (Functional). *Users must be able to add any file that are stored on their local machine into a container that is running. The container should recognize these files as if they are in that container's own file system.*

Req. 12 (Functional). *Users must be able to download any file stored within a running container and store it into user's own local file system.*

Req. 13 (Functional). *Users must be able to specify commands that will then be run by the container's operating system. The standard output of this process should be sent to the user as it is generated. While the program is running, if it searches for standard input, the user should be able to provide it as if it is running on their own local machine.*

Req. 14 (Functional). *Users must be able to stop a currently-running container on their system. This will terminate all of the RAM, CPU, and GPU usage of the container.*

Req. 15 (Functional). *Users must be able to delete a container that is downloaded on their system off of their system. Thus, this should free all of the space on their disk drive that was being used by the container.*

Req. 16 (Functional). *Users must be able to upload their own custom containers to the cloud (if they have the proper permissions to do so).*

Req. 17 (Performance). *The system must be able to install at least five containers, assuming the space allows for it.*

Req. 18 (Performance). *The system must allow at least five containers to run simultaneously, assuming memory allows for it.*

3.3 Container Storage

Req. 19 (Functional)**.** *Users can send a request to the container storage with a particular container name, and they should receive an archive containing a .qcow2 or .raw hard disk image and a .json config file.*

Req. 20 (Functional)**.** *Users can upload containers to the container storage by, after being asked for username/password authentication, sending an archive containing a .qcow2 or .raw hard disk image and a .json config file.*

3.4 Container Creation

Req. 21 (Functional)**.** *Users can create new containers using a special command that will come built with various options (featured in later requirements). Upon this command's termination, it will create a new container that can be utilized like other containers.*

Req. 22 (Functional)**.** *Users can specify the architecture that their container will use, which thus means that the machine can run binaries that are built for that architecture. The user should be able to use architectures including, but not limited to:*

- *SPARC*
- *ARM*
- *x86_64*
- *MIPS*

Req. 23 (Functional)**.** *Users can specify whether to include the standard development tools in Linux when they build a new container. This includes gcc and gdb.*

Req. 24 (Functional)**.** *Users can provide packages to be included and installed into their containers. These packages will be represented as .deb files (debian packages), and providing this during creating will install these tools into the container.*