Virtual Development Environment in a Box
Master Test Plan

Ian Orzel
Dylan McDougall

October 2022

Contents

Chapter 1

Introduction

1.1 Purpose

Previous documents have laid out a list of requirements for the project as well as the design for the project. This document will lay out the plan in order to ensure that the product of this product fulfills all requirements and has the proper design.

1.2 Scope

This document will describe the efforts that will be used to validate that the Virtual Development Environment in a Box project works as is intended. This document will list a series of tests along with expected outputs that the final Virtual Development Environment in a Box project must pass in order to be considered complete. We plan to test the pre-built images, the container storage, the container manager, and the container creator. Although we will not have tests for every possible input a user will attempt, these tests are made to ensure that each feature works properly and ensure that some common anomalies are dealt with properly.

1.3 System Overview and Key Features

The system that is being addressed by this document is the Virtual Development Environment in a Box system, and it consists of a couple of parts. It first consists of some pre-built images, which contain virtual Linux environments. Then, user's will have a container manager on their local system that is used for running and interacting with containers. User's will also have access to a container creator which allows users to create their own custom containers. Finally, there will be virtual container storage that allows for user's to download images to be put on their local system.

1.4 Test Overview

The tests described in this document will be used to validate that the system is operating as expected. The tests will be placed to focus on a particular component of the system, and there will be some tests that ensure that the entire system is working together as expected. For each test described in this document, the following will defined for each test:

- **Name**: The informal name of the test.
- **Identifier**: A unique identifier for the test.
- **Description**: A brief one to two sentence description of the test.
- **Preconditions**: A list of conditions that must be met before doing the test.
- **Methods**: A series of steps that will be done for the test.
- **Output**: The output that the system should give after completing the methods.
- **Postconditions**: A list of conditions that must be met after doing the test.

Chapter 2

Specific Tests

2.1 Pre-built Containers

2.1.1 Compiler Theory

Test 1 (C Compile and Debug):

Description: The image will be used to compile a basic C program that uses malloc using gcc and then have it debugged using gdb.
Preconditions: Qemu should be properly set up independently on the system and be able to run the container.
Methods:

1. Run qemu on the image from the terminal.
2. Transfer C program that runs malloc into the image.
3. Compile the C program using gcc on the image.
4. Run compiled binary to ensure that it runs without error.
5. Run gdb on the binary that is output.

Output: This should output the standard gdb output when used on a binary.
Postconditions: The image should now contain a SPARC compiled binary of the C program.

Test 2 (Java Compile):

Description: The image will compile and run a "Hello World" Java program.
Preconditions: Qumu should be properly set up independently on the system and be able to run the container.
Methods:

1. Run qemu on the image from the terminal.
2. Transfer "Hello World" Java program into the image.
3. Compile the Java program on the image.
4. Run the Java program on the image.

Output: This should output "Hello World".
Postconditions: The image should now contain a compiled Java program.

Test 3 (SPARC Binary):

Description: The image will run a basic pre-made SPARC binary.
Preconditions: Qemu should be properly set up independently on the system and be able to run the container.
Methods:

1. Run qemu on the image from the terminal.
2. Transfer the basic SPARC binary onto the image.
3. Run the SPARC binary on the image.

Output: It should output the expected output of the binary.
Postconditions: N/A

Test 4 (GNU Assembler and Linker):

Description: The image should assemble an assembly program which calls a C program which calls malloc.
Preconditions: Qemu should be properly set up independently on the system and be able to run the container.
Methods:

1. Run qemu on the image from the terminal.
2. Compile the C program into an object file using gcc.
3. Assembly the assembly program into a binary file using gas.
4. Link the assembly binary to the C object file using gld.
5. Execute the assembly binary.

Output: No notable output for this.
Postconditions: The assembly binary should successfully assemble, link, and should be executable.

2.1.2 Programming Language Concepts

Test 5 (Compiling and Running Programs):

Description: The image will compile and run programs written in the Go, Rust, Ada, and Haskell programming languages.
Preconditions: Qemu should be properly set up independently on the system and be able to run the container.
Methods:

1. Run qemu on the image from the terminal.
2. Transfer "Hello World" Go, Rust, Ada, and Haskell programs.
3. Compile the respective programs in each of their languages.
4. Run each program for each language.

Output: It should output "Hello World" for each of the programs.
Postconditions: The image should contain compiled binaries for each of the input programs.

2.2 Container Manager

Test 6 (Basic Container Workflow):

Description: This test consists of starting an image, running a basic command on the container, getting the output from that command, and then stopping the image.
Preconditions: The machine should have a basic container installed on it.
Methods:

1. Run start command on the image.
2. Run command to transfer file into container that just contains "Hello World".
3. Run command targeting image that runs "cat" of the file transferred on the container.
4. Run stop command on the image.

Output: The program should output "Hello World" when the command is run.
Postconditions: The image should now contain the file that was transferred in. Also, the container should be entirely stopped and not running on the system.

Test 7 (Downloading File From Container):

Description: The image should be able to download a file from the container.
Preconditions: A container must be currently running on the system.
Methods:

1. On the container, use echo and file redirection to create a file that has the content "Hello World".
2. Use the system to download this file from the image.

Output: No notable output for this.
Postconditions: The computer should now contain the created file on their file system.

Test 8 (Standard Input/Output):

Description: When running a program in the container, the user should be able to provide standard input to the program.
Preconditions: A container must be currently running on the system.
Methods:

1. On the container, run a binary that takes in standard input and echos it into standard output.
2. Provide "Hello World" into standard input.

Output: The program should output "Hello World".
Postconditions: N/A.

Test 9 (Delete Container):

Description: A container set up on the system should be able to be removed at any time.
Preconditions: A container must be installed on the system.
Methods:

1. Run the remove command targeting the container.

Output: No notable output from this process.
Postconditions: The container should no longer be installed on the system and should not be able to be accessed.

Test 10 (Download Container):

Description: A container must be able to be downloaded from the Internet.
Preconditions: A container is currently hosted in a repository for the system to access.
Methods:

1. Run command to download container for internet for a container currently being hosted.

2. Run the container.

3. Run the "whoami" command.

Output: The program should output the name of the user for the container.
Postconditions: The container hosted on the Internet should now be on the system.

Test 11 (Upload Container):

Description: A container on the local file system must be able to be uploaded to an Internet server assuming that they are properly authenticated.
Preconditions: A container must be on the local file system, a server must be set up, and the user must have the authentication for the server.
Methods:

1. Run the upload command on the container.

2. Properly provide the authentication for the server.

Output: No notable output.
Postconditions: The container should now be available on the server.

Test 12 (Multiple Containers):

Description: This test involves downloaded and running five containers at the same time.
Preconditions: The system should have no other containers installed and none running.
Methods:

1. Using the container creator, create five containers of any type.

2. Run each of these containers.

Output: No notable output.
Postconditions: Five containers are now installed, and all of them are running. Ensure that they are all properly installed and running.

2.3 Container Storage

Test 13 (Download From Storage):

Description: A container file should be able to be extracted from a storage location.
Preconditions: The storage should contain a container.
Methods:

1. Send a request to the server for a container.
2. Transfer the result from the request into the file.

Output: No notable output.
Postconditions: The computer should now have an archive that contains a .qcow2 file and .json file.

Test 14 (Upload to Repository):

Description: A container file should be able to be uploaded to a storage location.
Preconditions: The user should have access to the authentication for the server. Also, the user should have a container set up locally.
Methods:

1. Send a container upload request to the server.
2. Properly authenticate for the request.

Output: No notable output.
Postconditions: The server should now have the new container on it.

2.4 Container Creator

Test 15 (Create Containers of Various Archs):

Description: This test involves created a variety of containers that use different Architectures
Preconditions: Virtual Development Environment in a Box must be set up on the system.
Methods:

1. Follow the wizard to create images of architectures: SPARC, ARM, x86, and MIPS.
2. Start each container.
3. Run "uname -m" on each container.

Output: For each container, it should output the name of the architecture that is was created to have.
Postconditions: Four containers should now be created on the system.

Test 16 (Create Containers with Build Tools):

Description: This test involves creating a container that has standard build tools.
Preconditions: Virtual Development Environment in a Box must be set up.
Methods:

1. Follow the creation wizard to create a container that contains standard build tools.
2. Start the container.
3. Transfer a standard "Hello World" C program to the container.
4. Compile the C program and run it in the container.

Output: It should output "Hello World".
Postconditions: There should now be a new container on the system that contains a C program and a binary in its file system.

Test 17 (Add Custom Packages):

Description: This test involves creating a container that comes installed with a custom Debian package.
Preconditions: Virtual Development Environment in a Box must be set up.
Methods:

1. Download a .deb file containing Python 3.
2. Follow the creation wizard to create a container that uses the Python 3 .deb file.
3. Start the container that was created.
4. Transfer a standard "Hello World" Python 3 script into the container.
5. Run the script that was transferred.

Output: It should output "Hello World".
Postconditions: There should be a new container on the system that contains a Python program in its file system.