## Introduction

## Testbed Setup

### Machine Used

### Control for Outside Factors

In order to maintain a somewhat constant execution environment, factors that may influence the system were considered and controlled.

**Machine Specification**

Each instance of the experiment was executed singly on a lab pc, the hardware specifications of which were logged using the lscpu command. Variations in overall host processing power and memory availability between machines of different specifications can impact the resources allocated to containers due to CPU scheduling etc. and hence needed to be controlled for.

**Execution of other processes**

To ensure that the CPU and memory resource availability was not unintentionally limited due to other processes running on the system, the htop command was used before executing the containers to view and stop any non-essential system processes. This included processes that were initiated by other users.

### Setup Steps

To allow the containers to run correctly and to provide some data permanence within the ephemeral container file system, there are a number of prerequisite steps that must be taken before the containers can be run. This can be mostly ignored for anyone attempting to repeat the experiment as most of the setup and experimental steps are automated by coded shell scripts. These scripts handle resetting the container engine system, building images, running and removing containers, recording stats to files, retrieving data from within the container etc. allowing the experiment to be run by invoking the init.sh script with the desired options set.

#### init.sh

init.sh is the experiment initialisation script that should be called by the user. From an abstracted perspective, calling the script with its options is the only setup step required.

##### Script Usage and Options

**Invocation**

bash /path/to/init.sh [options]

**Options**

-t : Flag to indicate whether time-series data should be collected, with container statistics sampled each second

-n program-name: Option to specify that only the program program-name in the directory /cs/studres/CS1007/Coursework/A03/programs/ is to be tested. If omitted, all programs in this directory will be used.

Although not explicitly indicated in the practical specification, the scripts provides options for the user to collect data recording the execution time of the containers for varying CPU resource limits. The aim of this facility being to provide a more informed answer to question posed in the specification: “Would [the programs] benefit more from running on a machine with more resources?”. The following options, if included, will enable this functionality.

-p peak-CPU: Option to specify the upper limit of CPU allocation to be tested

-m mean-CPU: Option to specify the lower limit of CPU allocation to be tested. Denoted as the mean-CPU as this is the value (or slightly below) that is used to collect the data discussed in this report. Setting the lower limit to be substantially less than the mean utilised by the program/container when run without limit is unlikely to yield meaningful insight into the CPU usage by the execution time metric.

##### Script Function

Other steps that must be taken to allow the containers to function properly and reliably are automated by the script. The script yields a testbed container within which nested containers to run the programs are built, launched, and managed. Containerizing the testbed allowed for easier control over and facilitated isolation of the nested containers in which the programs are run.

**Container Engine Reset**

Before attempting to run the testbed container, the script will reset the container engine (podman) system to remove previous containers, images, volumes etc. This is necessary due to the flow of variable command line data into the testbed in order to configure the experiment. The testbed image must therefore be rebuilt to contain the values of the option variables for this invocation of the script; in order to build a testbed image with the same tag specified (such that it can be more clearly referenced elsewhere in the script), it must be ensured that previous image builds are removed to prevent these being used to run the testbed container. A similar logic applies to removing old container and volume instances as these are also referenced explicitly by their tags/names and hence could cause conflicts with existing instances. Although the podman container engine can allocate randomly generated, unique names to its members in the absence of a specified name; the script would then need to acquire and store this name (or, equivalently, the member id). Resetting the system to ensure a clean execution environment seemed a cleaner way to handle this. The user is prompted to confirm the reset in the event that this is not acceptable.

**Volume Generation**

Under normal circumstances, container file systems are ephemeral and hence any data generated within the container will be lost once the container is terminated. Without solution, this would prove to be problematic for collecting experiment data from a testbed running within a container. The practice of attaching a container volume, which is a portion of the host file system mounted onto the container such that it is accessible from within the container, allows for container data to persist on the host. A volume for the parent testbed container is created and mounted by the script allowing for data to be passed from the host into the testbed and vice versa. Data into the testbed from the host include the program files to be tested as well as testbed.sh script and program\_containerfile container file which must be available to allow for images to be built of containers responsible for running the programs. The testbed data needing to persist on the host are the statistic files generated when the program containers are run.

**Testbed Container Launch**

The main responsibility of the init.sh is to build and run the testbed container using the testbed\_containerfile discussed further below. As the testbed container must itself be able to create new containers, it must be run in a privileged mode.

#### testbed\_containerfile

The testbed image is based on the almalinux parent image. The image is extended by installing additional utilities required by the testbed, namely podman, procps, and bc. When containers built from this image are run, the testbed.sh script is run automatically with any options specified.

#### testbed.sh

This script is run automatically within the testbed container and is responsible for building and running the containers to execute the provided programs.

For each of the programs to be tested, the script creates a directory in which the results of the experiment (podman stats reports, execution time records, host specification logs etcl) can be written to and retrieved from. An image is built from which a container to run the program is generated; a volume containing a blank file accepted as an argument by the programs to modify is created to be mounted onto these container instances.

The behaviour of the script following this initial setup is largely dependent on the experiment parameters supplied by the user.

**Time Series Data**

**As described in the practical specification, the script provides the option to record data at one second time intervals using the podman stats command for a single container running a particular program. This data is redirected to a csv file and stored in the results directory created within the testbed container file system to be retrieved at the end of the experiment. The data is collected using the podman stats command as discussed further below.**

**CPU Limited Execution Time Data**

**If indicated, the script will also run the programs in containers with varying limits of CPU utilisation. Using the podman run –cpus option, the number of CPUs available to be used by the container processes is limited**

## Method

### Podman Stats

As recommended in the practical specification, the podman stats command is used to access the container’s resource usage statistics. By default, the command displays a live stream resource use statistics report every 5 seconds for the running containers. In order to produce the required second-by-second usage data for a particular container executing one of the provided programs, the report interval must be modified and the container to provide statistics for must be specified. The report interval can be set using the --interval=seconds option, where seconds is the time in seconds between stats reports, which, for the purposes of this practical, should be set to 1. In order to display statistics for only one container, the container’s name can be provided as an argument to the command. The container name can be set within the podman run command using the --name=container\_name option, this name is then used to limit the statistics report to that single container and not all running containers. To filter the statistics report to include only the data pertaining to CPU utilisation and memory usage, the –format=template option with a Go template containing the relevant placeholders is added. To record this second by second data stream, the output of the command is redirected to a csv file, an output to standard error (which should not be recorded in the csv file) is redirected elsewhere. In its entirety, the command used to produce the podman stats report is:

podman stats --format "{{.CPUNano}},{{.CPU}},{{.AvgCPU}},{{.MemPerc}}" --interval=1 "$container\_name" 1>> "$stats\_file\_path" 2>/dev/null

where $container\_name, $stats\_file\_path are script variables set appropriately for the current container execution.

### Control for Random Error

Inevitably, results will be to some extent impacted by random errors that occur during the experiment due to fluctuations in the system. In attempt to reduce the impact of these errors, the containers executing each program were run with (as much as possible) identical setups. The time-series data for each of these replicants was collected and examined for outliers which were removed. The time-series data could then be aggregated across runs by taking the mean values of the statistic set each second.

## Results

## Conclusion