WAVE User Manual

Leandro Cavalcanti de Almeida
(leandro.almeida@ifpb.edu.br)
Paulo Ditarso Maciel Jr.
(paulo.maciel@ifpb.edu.br)
Danilo Cavalcante Beuttenmuller
(danilo.cavalcante@academico.ifpb.edu.br)
Matheus Faelson de Almeida Valério
(matheus.faelson@academico.ifpb.edu.br)
Caio Luiz Lacerda Terto Silva
(caio.luiz@academico.ifpb.edu.br)
Icaro Machado da Silva
(icaro.silva@academico.ifpb.edu.br)

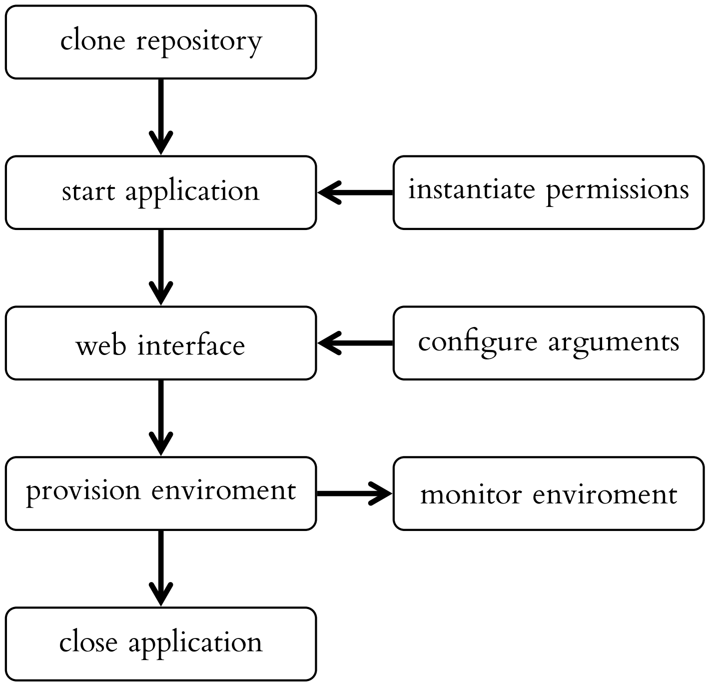
https://github.com/ifpb/new_wave
version 1.1, March 2, 2025

Abstract

This manual presents the WAVE tool, a workload generator for verifiable experiments. Being application-agnostic, WAVE has been extensively used in load testing for video-on-demand applications and key-value storage. In its current version, WAVE can generate workloads for three distinct patterns: sinusoidal, *flashcrowd* and *step*. Additionally, it is possible to toggle the use of a micro-burst generator on the network.

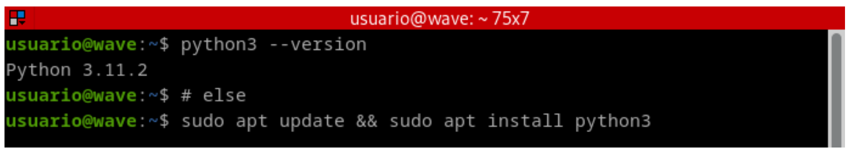
Contents

Quick Workflow Guide



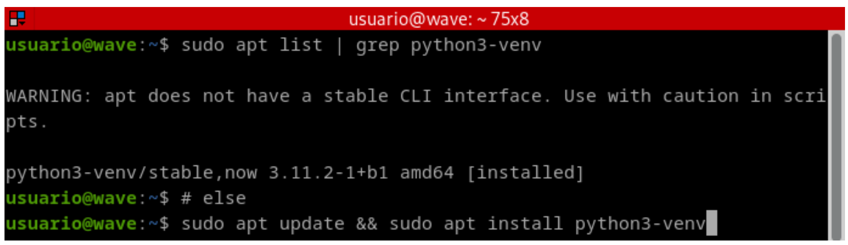
1 Checking the Required Requirements

1.1 Checking if **`Python3`** [8] is installed and it's version:



```
usuario@wave: ~ 75x7
usuario@wave:~$ python3 --version
Python 3.11.2
usuario@wave:~$ # else
usuario@wave:~$ sudo apt update && sudo apt install python3
```

1.2 Additionally, the **`VirtualEnv`** virtual environment is required :



```
usuario@wave: ~ 75x8
usuario@wave:~$ sudo apt list | grep python3-venv

WARNING: apt does not have a stable CLI interface. Use with caution in scri
pts.

python3-venv/stable,now 3.11.2-1+b1 amd64 [installed]
usuario@wave:~$ # else
usuario@wave:~$ sudo apt update && sudo apt install python3-venv
```

1.3 Checking the **`Docker`** [4] and **`docker compose`** components:

```
usuario@wave: ~ 75x6
usuario@wave:~$ docker --version
Docker version 27.5.1, build 9f9e405
usuario@wave:~$ # else
usuario@wave:~$ sudo apt update && sudo apt-get install docker-ce docker-ce
-cli containerd.io docker-buildx-plugin
```

```
usuario@wave: ~ 75x6
usuario@wave:~$ docker compose version
Docker Compose version v2.32.4
usuario@wave:~$ # else
usuario@wave:~$ sudo apt install docker-compose-plugin
```

1.4 Checking what version of **VirtualBox** [11] is installed:

```
usuario@wave: ~ 75x6
usuario@wave:~$ vboxmanage --version
7.1.6r167084
usuario@wave:~$ sudo apt install virtualbox
```

1.5 Checking what version of **Vagrant** [10] is installed:

```
usuario@wave: ~ 75x6
usuario@wave:~$ vagrant --version
Vagrant 2.3.4
usuario@wave:~$ # else
usuario@wave:~$ sudo apt install vagrant
```

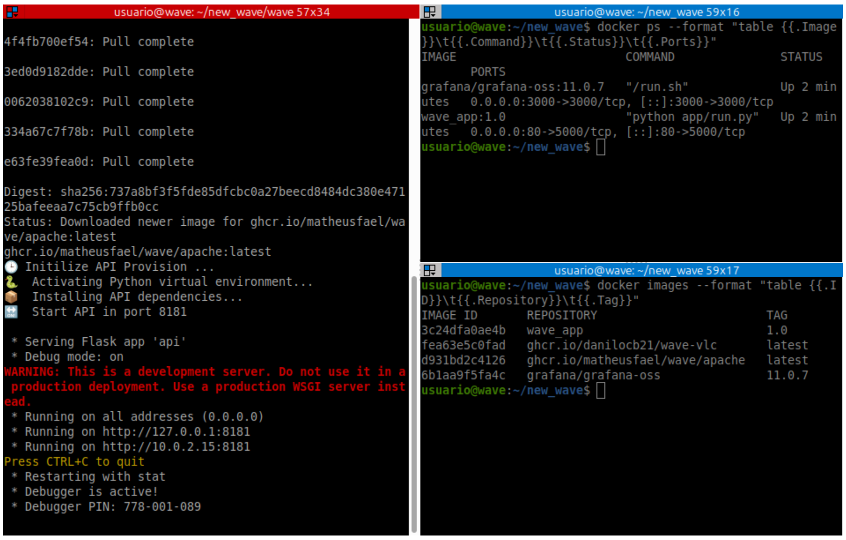
The versions shown in the figures for steps 1.1-1.5 were those tested at the time of this manual's creation.

2 Downloading the Code and Starting the Environment

2.1 Cloning the official repository and starting the system:

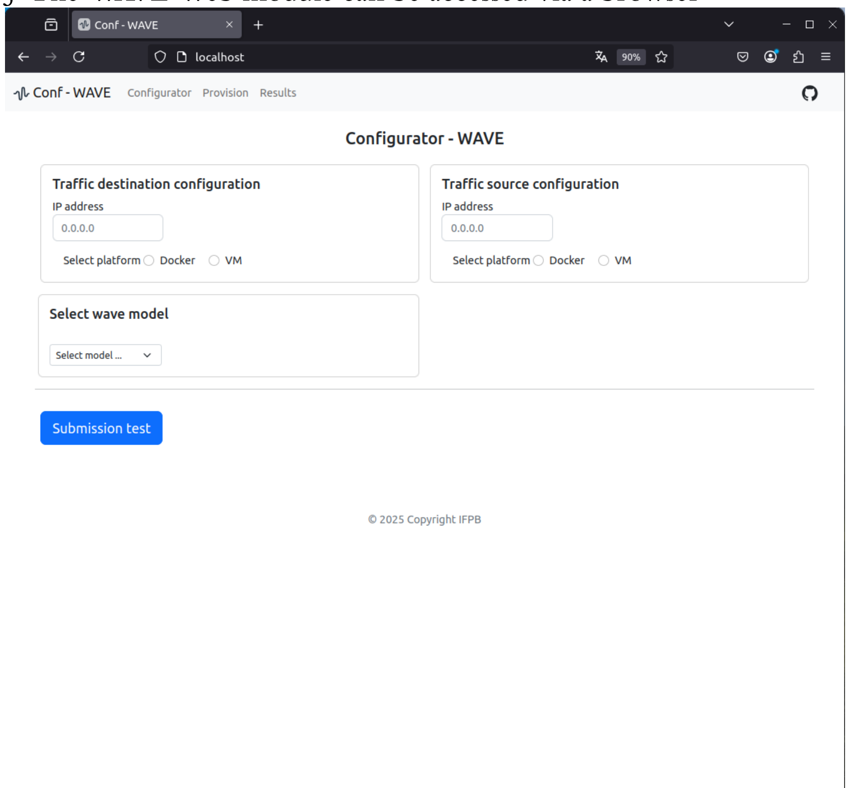
```
$ git clone https://github.com/ifpb/new_wave.git
$ cd new_wave/wave
$ ./app-compose.sh --start
```

2.2 Checking the execution in a **Docker** environment:



As can be seen in the figure above, the WAVE Initialization module uses two containers for its execution: `wave_app` and `grafana-oss`. On the left side of the figure, we have the output of the WAVE startup command, which corresponds to the output of line 3 of the code shown in section 2.1.

2.3 The WAVE Web module can be accessed via a browser



The form contains fields for entering network data for both the traffic load source and destination. In addition to the IP address, it is possible to select environment provisioning through a container or a virtual machine with configurable memory size and number of virtual CPUs. Finally, the user can choose which workload model to apply, either *sinusoid*, *flashcrowd* or *step* and if they want to use micro-burst as well.

2.4 Virtualization environment started by the Provisioning module:

The virtual machines used as the source and destination of the traffic load are configured from the `app/provision` directory. For this purpose, the *Vagrant* tool and the *VirtualBox* hypervisor are instantiated to run the environment. This directory contains a pre-edited `Vagrantfile`, whose configurations are provided by another file called `config.yaml`. The latter is generated after filling out the form in the WAVE WEB module, containing the arguments assigned by the user. Additionally, two pre-configured `vagrant boxes` (`wave-server` and `wave-client`) have been set up with the necessary software for generating load between the source and destination (`IPerf` [6] and traffic load models), respectively referred to as Client VM and Server VM. If the `boxes` are not yet installed on the local machine, they will be downloaded during the system's first execution. Below is the `Vagrantfile` code responsible for provisioning the environment.

```ruby
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.require_version '>= 1.6.0' # Minimum Vagrant version
VAGRANTFILE_API_VERSION = '2' # Vagrant API version

require 'yaml' # Require 'yaml' module

# Edit config.yml to change VM configuration details
machines = YAML.load_file(File.join(File.dirname(__FILE__),
↪  'config.yaml'))

$SCRIPT = <<-EOF
server_ip=$(grep 'ip:' /vagrant/config.yaml | cut -d: -f2 |
↪  head -n1 | sed -e 's/\"//g')
echo -e "$server_ip server" | sudo tee -a /etc/hosts
client_ip=$(grep 'ip:' /vagrant/config.yaml | cut -d: -f2 |
↪  tail -n1 | sed -e 's/\"//g')
echo -e "$client_ip client" | sudo tee -a /etc/hosts
EOF

# Create boxes
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|

# Iterate through entries in YAML file to create VMs
machines.each do |machines|
```

```

  # Configure the VMs per details in config.yml
  config.vm.define machines['traffic'] do |set|

    # Specify the hostname of the VM
    set.vm.hostname = machines['traffic']

    if machines['traffic'] == "server"
      set.vm.box = "ifpb/wave-server" # VM Server box
      set.vm.hostname = "wave-server" # VM Server hostname
    else
      set.vm.box = "wave/client" # VM Client box
      set.vm.hostname = "wave-client" # VM Client hostname
    end

    # Iterate through networks as per settings in machines
    set.vm.network "private_network", ip: machines['ip']

    set.vm.provider 'virtualbox' do |vb|
      vb.memory = machines['ram'] # Configure RAM
      vb.cpus = machines['vcpu'] # Configure CPU
    end # set.vm.provider 'virtualbox'

    set.vm.provision "shell", inline: $SCRIPT

  end # config.vm.define
end # machines.each
end # Vagrant.configure
```

In the context of the WAVE project, Docker is employed as a portable, lightweight, and scalable solution for containerized deployment of the application's core services. This approach significantly streamlines the environment setup, as Docker containers are utilized to encapsulate both the client and server components in isolated environments. The use of Docker further facilitates the rapid replication of experimental setups across different systems, ensuring a high degree of consistency and minimizing the time required for environment configuration.

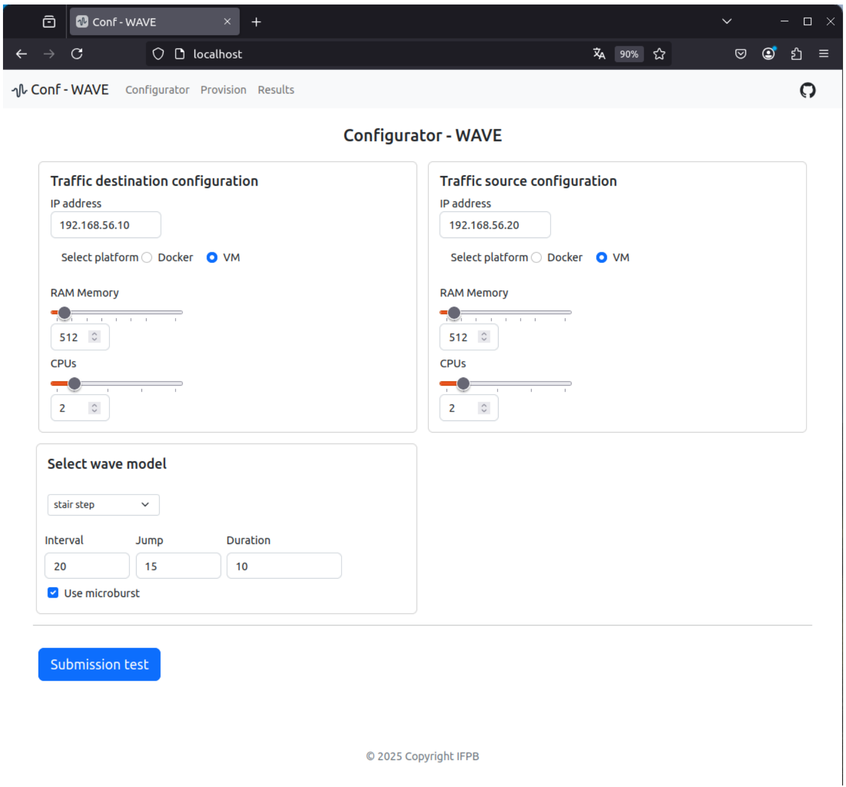
For the provisioning of the environment, Docker Compose is employed to launch the containers automatically as soon as the user initiates the provisioning process. Subsequently, the client container, which runs VLC Media Player (VLC), remains in a waiting state, ready to receive commands through the web interface. The server container, in turn, is configured with Apache HTTP Server (Apache) to host the video content. Docker Compose serves as an orchestration tool, enabling the definition and management of multi-container applications. The various services that comprise WAVE—specifically the client and server components—are articulated and managed within isolated containers. This approach not only simplifies the orchestration of these services but also ensures the rapid deployment and scalability of the environment. Below is the docker-compose.yaml file responsible for provisioning the environment.

```yaml
services:
  apache:
    image: ghcr.io/ifpb/new_wave/wave-apache
    container_name: server
    #ports:
    #- "80:80"

  client_container:
    image: ghcr.io/ifpb/new_wave/wave-vlc
    container_name: client
    privileged: true
    #environment:
      #- DISPLAY=${DISPLAY}
    volumes:
      - /etc/localtime:/etc/localtime:ro
      - ./logs:/home/vlc/logs
      - /tmp/.X11-unix:/tmp/.X11-unix
    depends_on:
      - apache
```

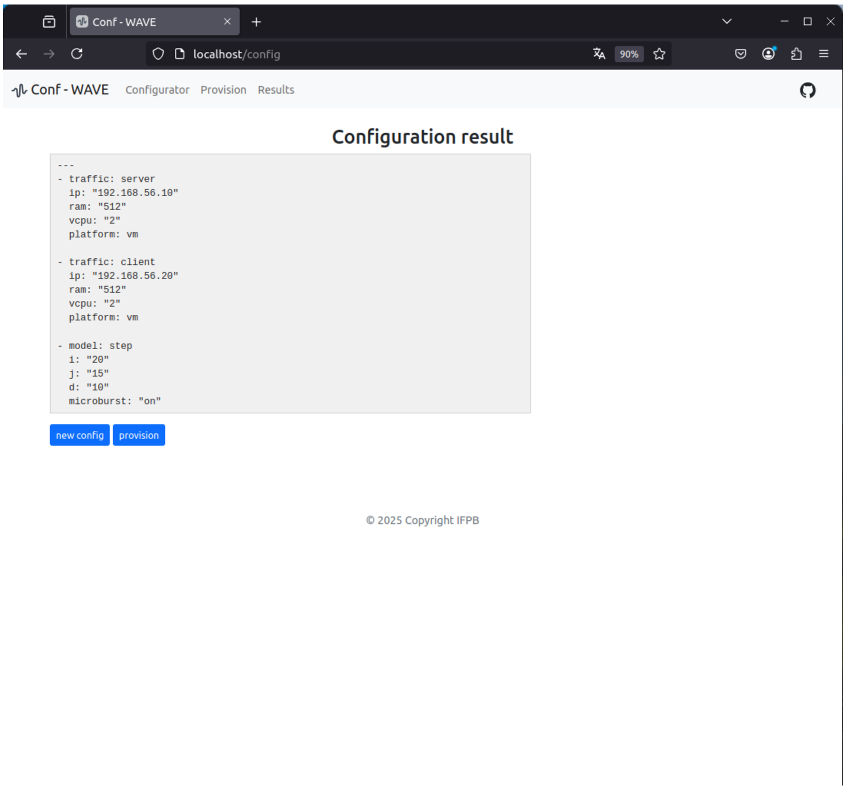
3 Interacting with WAVE WEB

3.1 Once the system is initialized, the user must enter the arguments:



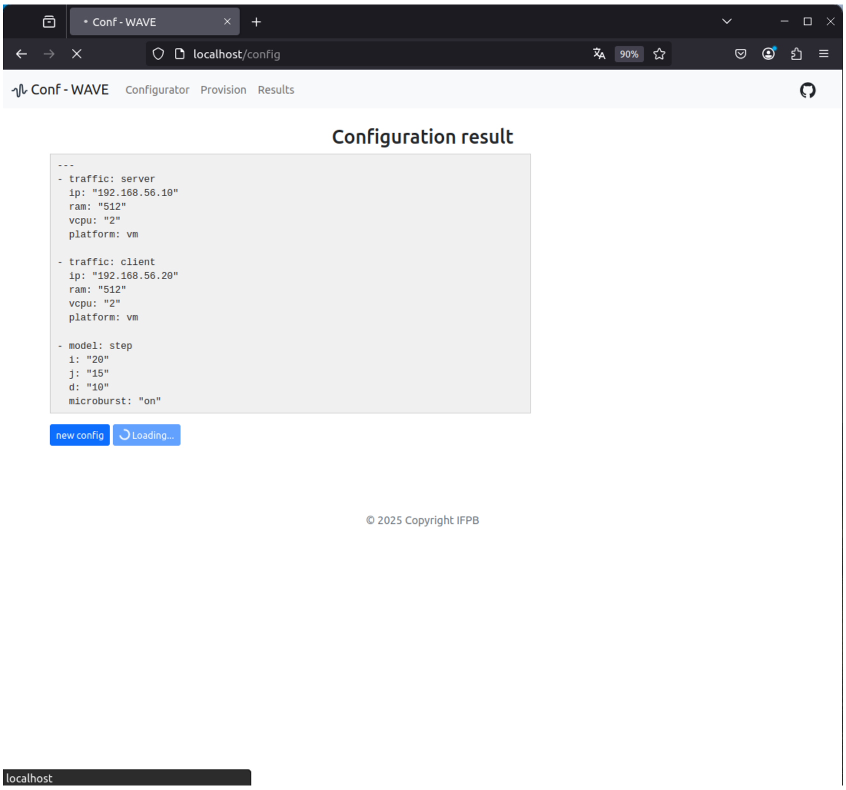
The previous figure illustrates the completion of the web form with the necessary parameters for executing a workload based on a *sinusoid* model. First, the IP addresses of the Server VM and Client VM are configured. Attention to an important step in VM configuration: since the environment is set up to use virtual network interfaces in *private* mode (line 40 of the `Vagrantfile` above) or *host-only* mode in `VirtualBox`, `Vagrant` requires these interfaces to be configured within the `192.168.56.0/24` address range. To use interfaces in *bridge* mode, simply change the term *private* to *public* in the `Vagrantfile`. In addition to addressing, values were assigned for the amount of virtual memory and virtual CPUs in each VM. A *sinusoid* load model is instantiated at the bottom of the form using its respective arguments. For a better understanding of the supported load models, it is recommended to read [1, 3, 9].

3.2 Consolidation of inserted arguments:



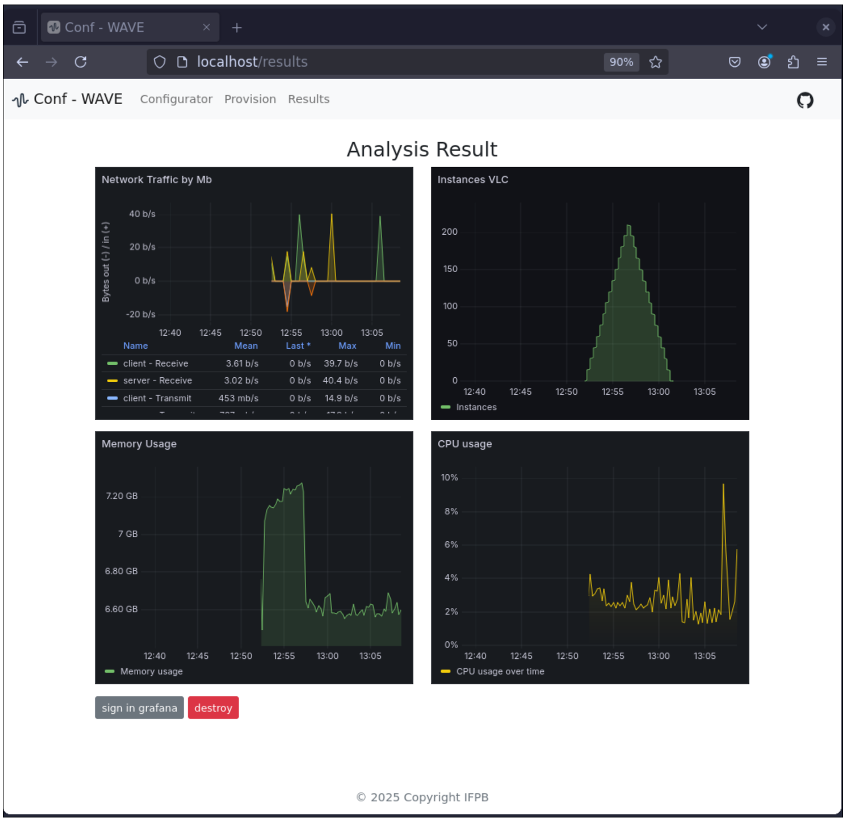
After entering the arguments in the homepage form, WAVE WEB displays a confirmation page for the input data. These arguments are edited in a YAML-formatted file called `config.yaml`. If the user notices any errors in the entered arguments, they can return to the homepage using the new configuration button or the menu in the top bar of the web page. Once the data has been verified, the user can proceed with the environment provisioning.
It is worth noting that the Provisioning module, responsible for executing the environment, automatically configures and starts the two VMs involved in generating and receiving the traffic load. More specifically, the Server VM is launched with `IPerf` already running in server mode, ready to receive traffic on its default port (5201). The Client VM, in turn, is instantiated with `IPerf` pre-installed, along with the `Python` code responsible for the *sinusoid*, *flashcrowd* and *step* load models. Additionally, one of the load models is automatically instantiated based on the parameters entered by the user, and the specific workload (traffic) is sent to the Server VM. Depending on the selected model, `IPerf` instances in client-mode are started and terminated to emulate the desired behavior for the given load model. On the Client VM, the number of running instances can be verified through the recorded *log* files. Furthermore, the Monitoring module allows users to track various execution metrics for the Server VM, as detailed below.

When Docker is used, the provisioning process operates in a similar way to the virtual machine setup. In this scenario, the Server



container is launched with `Apache` [2] pre-installed and configured to serve video content via HTTP. Apache is set up to provide the necessary media files to the Client container, which is instantiated with `VLC` [12] (VideoLAN Client) pre-installed. `VLC` runs in a client mode, where it is configured to request and stream the video content served by the Apache server. Upon initialization, one of the video load models is automatically selected based on user input. This determines the behavior of the video traffic, such as varying the number of concurrent video requests or simulating sudden bursts in demand.

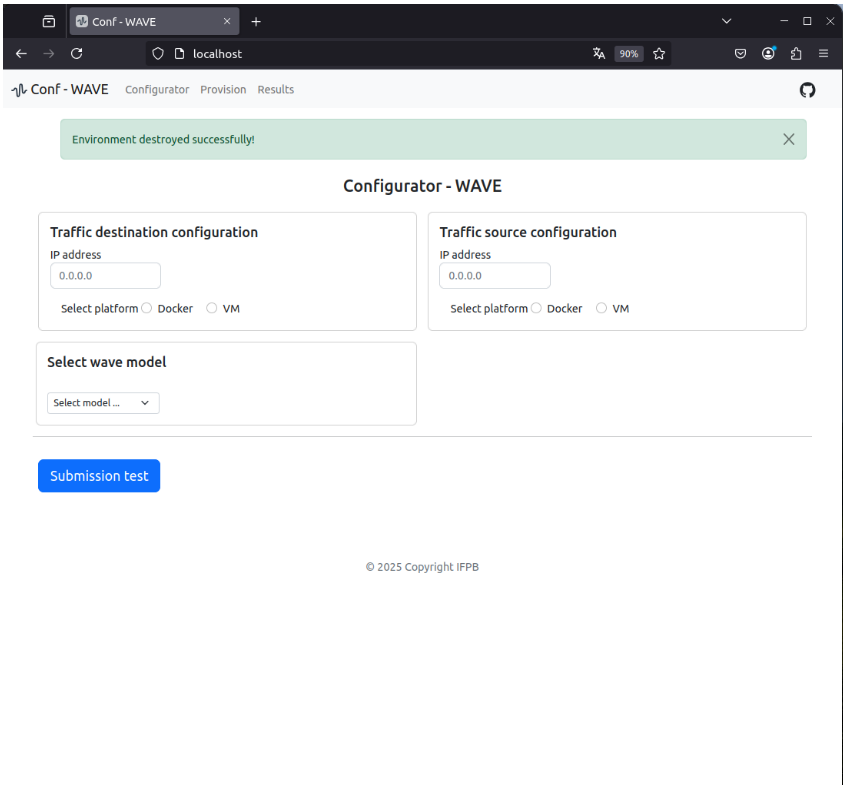
3.3 Once the environment is provisioned, a results page follows:



The Monitoring module is responsible for collecting and displaying execution metrics from the Server VM. For this purpose, the server's `vagrant box` is preconfigured with the `Prometheus` [7] and `node_exporter` tools. Using `Prometheus` as a data source, the `Grafana` container can display all metrics collected by `node_exporter`, including various indicators related to CPU, memory, and disk usage, as well as network traffic transmission and reception. After VM provisioning, the page above displays the rate of bytes received and sent through the Server VM's network interfaces. This enables, for example, real-time monitoring of the experiment, which involves sending traffic between the source and destination according to a specific load model. Futhermore, it's also possible to see the data used by the container via the cadVisor container that starts with the web app. Additionally, users can open a new web page with the `Grafana` interface[1] [5] to explore other relevant metrics or even configure new custom dashboards.

3.4 At any moment, it's possible to close the containers/virtual machines:

[1] For the first login, the default username and password are `admin`.



By clicking the `destroy` button on the traffic verification page, the user is redirected to the homepage, where they can restart the experiment if desired.

4 Ending the WAVE Execution

4.1 Finalizing and removing the container environment:

```
$ ./app-compose.sh --destroy
```

By running the command above, the user terminates the WAVE WEB module and removes the containers responsible for the other initiated modules. To restart the entire system, simply execute the same command, replacing the `-destroy` argument with `-start`, as indicated in line 3 of the code in section 2.1.

References

[1] Leandro Almeida, Fábio Verdi, and Rafael Pasquini. Estimando métricas de serviço através de in-band network telemetry. In *Anais do XXXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 252–265, Porto Alegre, RS, Brasil, 2021. SBC.

[2] Apache - HTTP Server Project. `https://httpd.apache.org/`.

[3] Ismail Ari, Bo Hong, Ethan Miller, Scott Brandt, and Darrell Long. Managing flash crowds on the internet. In *MASCOTS 2003*, pages 246– 249, 11 2003.

[4] Docker: Accelerated, Containerized Application Development. `https://www.docker.com/`.

[5] Grafana: The open observability platform | Grafana Labs. `https://grafana.com/`.

[6] IPerf - the ultimate speed test tool for TCP, UDP and SCTPTEST the limits of your network + internet neutrality test. `https://iperf.fr/`.

[7] Prometheus - Monitoring system & time series database. `https://prometheus.io/`.

[8] Welcome to Python.org. `https://www.python.org/`.

[9] Rolf Stadler, Rafael Pasquini, and Viktoria Fodor. Learning from network device statistics. *J. Netw. Syst. Manag.*, 25(4):672–698, 2017.

[10] Vagrant by HashiCorp. `https://www.vagrantup.com/`.

[11] Oracle VM VirtualBox. `https://www.virtualbox.org/`.

[12] VLC Media Player. `https://www.videolan.org/vlc/`.