National College of Ireland

Configuration Manual

MSc Research Project
MSc in Cybersecurity

Abhai Panichiyil
Student ID: 23207167

School of Computing
National College of Ireland

Supervisor: Joel Aleburu

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Abhai Panichiyil
Student ID:	x23207167
Programme:	MSc. Cybersecurity **Year:** 2024-2025
Module:	Practicum
Supervisor:	Joel Aleburu
Submission Due Date:	15-09-2025
Project Title:	Implementing Zero Trust Security in Multi-Cloud and Hybrid Cloud Environments: Ensuring Consistent Identity Verification, Micro-Segmentation, and Secure Inter-Cloud Communication.
Word Count:	1296 **Page Count:** 11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Abhai Panichiyil

Date: 15-09-2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Introduction

This manual outlines the step-by-step configuration process for setting up a Zero Trust Architecture (ZTA) environment using Minikube-based Kubernetes clusters, Calico microsegmentation and VPN-secured hybrid connectivity. Each step is illustrated with screenshots and commands to ensure accuracy and reproducibility.

Environment Setup

Virtual Machine Deployment

- **Platform**: VirtualBox
- **VMs**:
    - **Controller Node** – Runs Minikube cluster and Calico
    - **Worker Node** – Hosts application pods
    - **VPN Gateway** – Establishes IPSec tunnel to simulate hybrid connectivity

Implementation

We started by deploying a Kubernetes cluster in Minikube using the Docker driver. After the cluster initialization, Calico was installed as CNI for enabling microsegmentation capabilities. We checked the kube-system namespace to verify that the Calico components (calico-node, calico-kube-controllers, calico-typha) were all running and healthy (kube-system pod calico).

This was followed by two Nginx deployments for the backend and the web app. Calico Network Policies were applied for network segmentation: Allow_A_to_B.yaml allowed traffic among certain backend pods; Deny_to_C.yaml disabled access to certain pods. Using some kubectl commands, these policies were verified to have the correct labels and selectors.

We checked connectivity, then executed interactive shell inside of the respective backend pod and ICMP ping tests — Allowed traffic was able to pass and restricted communication was blocked — Result: as expected.

Last but not least, we also measured latency with and without VPN. As previously mentioned, we expect that VPNs will add ingress and egress encryption overhead, as well as some routing overhead: the average round-trip time increased from just over 233 ms to 349 ms with VPN

usage (Table 1). In this post, we demonstrated an end-to-end setup that deploys, configures, and validates K8s microsegmentation with Calico.

Steps to be followed by a User

Check Nginx Installation

Confirm that Nginx is installed and running by accessing the default welcome page in the terminal output. If you see “Welcome to nginx!” along with basic setup instructions, the server is working correctly and ready for further configuration.

```
abhai@abhai-VirtualBox:~$ curl localhost
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

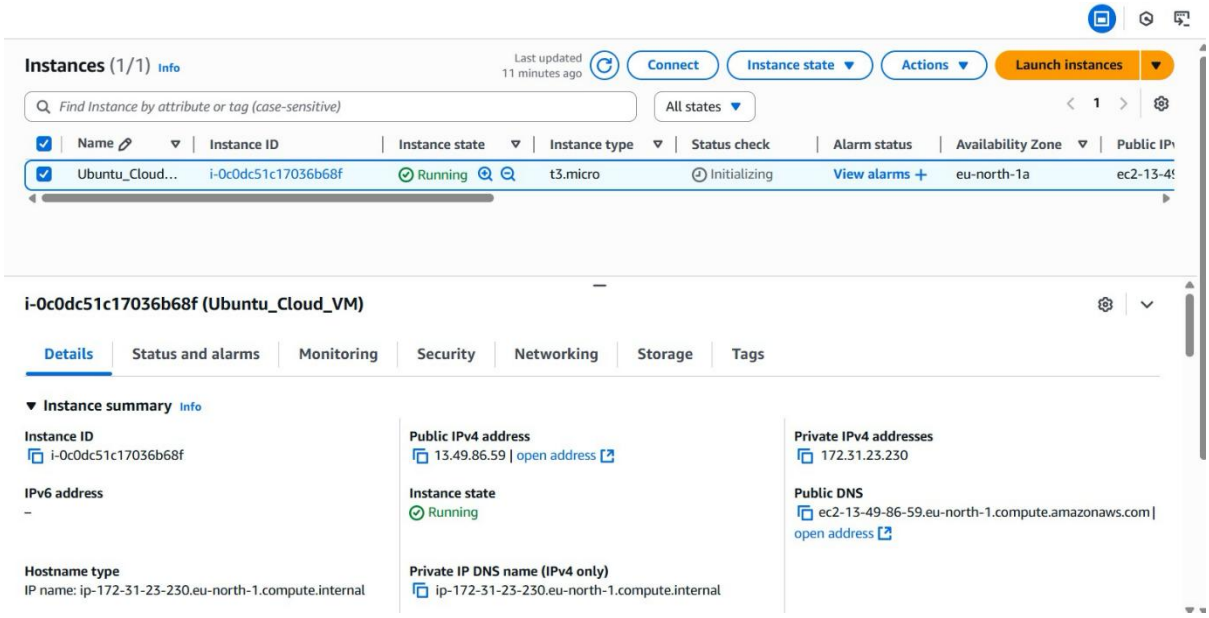
Configuration Flow for Cloud VM Access and Security

AWS EC2 Instance Details

This screen shows the **AWS EC2 Management Console** with an instance named **Ubuntu_Cloud_VM**.
Key details:

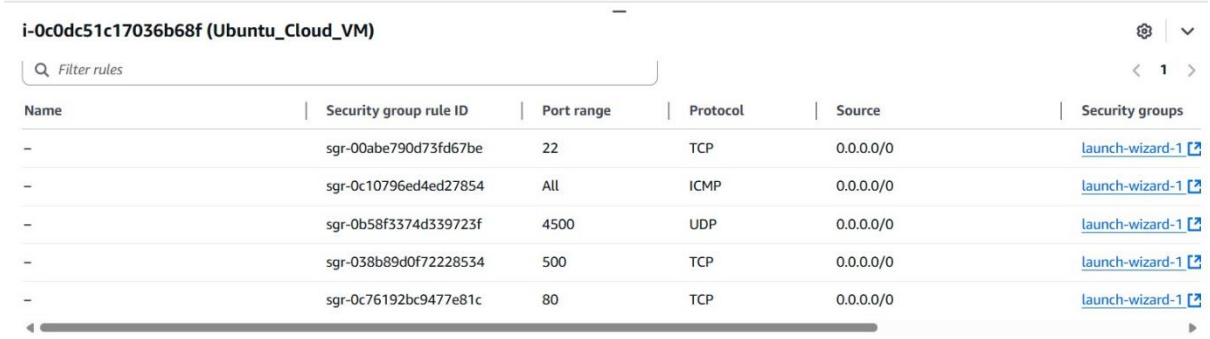
- **Instance ID**: i-0c0dc51c17036b68f
- **Instance Type**: t3.micro
- **State**: Running
- **Public IPv4 Address**: 13.49.86.59 (used for external SSH or HTTP access)
- **Private IPv4 Address**: 172.31.23.230 (used inside AWS VPC)
- **Public DNS**: ec2-13-49-86-59.eu-north-1.compute.amazonaws.com

Purpose:
This information is required to connect to the VM from your local system or another cloud environment.



Verifying VM Security Group Rules

- Once signed in, navigate to the **Security Groups** associated with your VM instance (e.g., Ubuntu_Cloud_VM).
- Ensure the inbound rules are properly set for required services:
  - **TCP Port 22**: For SSH access.
  - **ICMP**: For ping and connectivity checks.
  - **UDP Port 4500** and **TCP Port 500**: For VPN and secure tunneling.
  - **TCP Port 80**: For HTTP (web server) access.



Local Ubuntu VM System & Network Info

From the **local VirtualBox Ubuntu VM**, two commands were run:

- **lsb_release -a** → Displays OS details (Ubuntu 25.04, codename plucky).

- **ifconfig** → Shows active network interfaces:
  - **enp0s3**: Main Ethernet interface
    - IP Address: 10.0.2.15 (private/local network for VirtualBox)
    - MAC Address: 08:00:27:28:a6:b5
  - **lo**: Loopback interface (127.0.0.1)

Purpose:
Confirms the local VM is running and identifies the internal IP used for communication with AWS VPN.

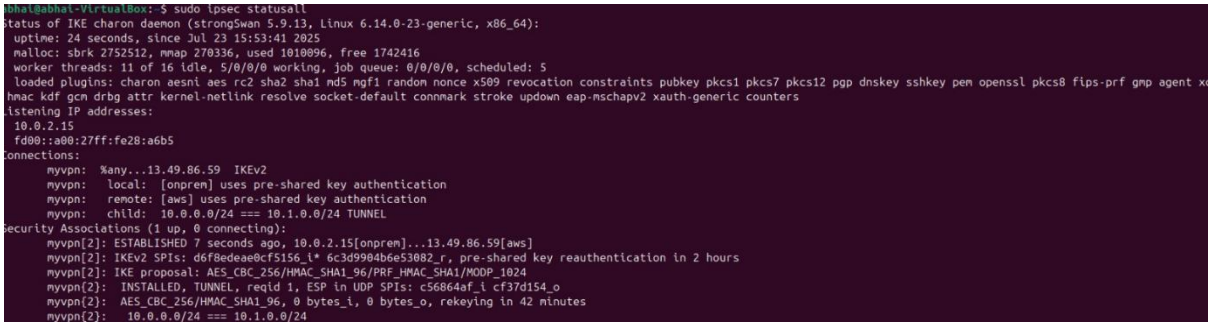
```
abhai@abhai-VirtualBox:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 25.04
Release:        25.04
Codename:       plucky
abhai@abhai-VirtualBox:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fd00::ad76:1464:94b3:c360  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::a00:27ff:fe28:a6b5  prefixlen 64  scopeid 0x20<link>
        inet6 fd00::a00:27ff:fe28:a6b5  prefixlen 64  scopeid 0x0<global>
        ether 08:00:27:28:a6:b5  txqueuelen 1000  (Ethernet)
        RX packets 424  bytes 365837 (365.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 310  bytes 38568 (38.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 230  bytes 18868 (18.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 230  bytes 18868 (18.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

VPN Connection Status (StrongSwan)

The **sudo ipsec statusall** command shows:

- **VPN Protocol**: IKEv2
- **Tunnel Established**:
  - Local (onprem): 10.0.2.15
  - AWS (remote): 13.49.86.59
- **Security Associations (SAs)**:
  - AES256 encryption
  - SHA1/SHA256 authentication
  - Subnets connected: 10.0.0.0/24 ↔ 10.1.0.0/24
- **Rekeying**: Set to refresh in 42 minutes

Purpose:
Confirms secure connectivity between the local VirtualBox VM and AWS EC2 VM through a working IPSec VPN tunnel.



- AWS Console (Instance Info) → **Identifies AWS VM details** (public/private IPs, DNS)
- Local Ubuntu VM Network Info → **Verifies local system's internal IP and readiness**
- VPN Status → **Shows successful encrypted tunnel between local VM and AWS instance**

Now Microsegmentation

Configuring Calico Microsegmentation in Minikube

Step 1: Start Minikube with Docker Driver

Begin by creating a Minikube cluster using the Docker driver. This ensures a containerized Kubernetes environment suitable for testing network segmentation. During initialization, the storage-provisioner is set up, and namespaces are prepared.

Once Minikube is running, create two deployments:

- **Backend** (nginx image)
- **Web** (nginx image)

  Both deployments will be used to validate network policy rules.

‘

```
abhai@abhai-VirtualBox:~$ kubectl get pods -n kube-system
NAME                                       READY   STATUS    RESTARTS      AGE
calico-kube-controllers-75cd4cc5b9-ds68l   1/1     Running   0             5m53s
calico-node-l4rq4                          1/1     Running   0             5m54s
calico-typha-75689d7846-f9kfp              1/1     Running   0             5m53s
coredns-674b8bbfcf-4md48                   1/1     Running   0             61m
etcd-minikube                              1/1     Running   0             61m
kube-apiserver-minikube                    1/1     Running   0             61m
kube-controller-manager-minikube           1/1     Running   0             61m
kube-proxy-xrvdq                           1/1     Running   0             61m
kube-scheduler-minikube                    1/1     Running   0             61m
storage-provisioner                        1/1     Running   1 (60m ago)   61m
abhai@abhai-VirtualBox:~$
```

Step 2: Verify Calico System Pods

After Minikube is ready, check the kube-system namespace to ensure Calico components are running:

- calico-kube-controllers
- calico-node
- calico-typha

All pods should show a **READY** status. Calico here acts as the Container Network Interface (CNI) plugin, enabling advanced network policies.

```
abhai@abhai-VirtualBox:~$ nano Allow_A_to_B.yaml
abhai@abhai-VirtualBox:~$ nano Deny_to_C.yaml
abhai@abhai-VirtualBox:~$ kubectl apply -f Allow_A_to_B.yaml
networkpolicy.networking.k8s.io/allow-a-to-b created
abhai@abhai-VirtualBox:~$ kubectl apply -f Deny_to_C.yaml
networkpolicy.networking.k8s.io/deny-to-c created
abhai@abhai-VirtualBox:~$ kubectl get networkpolicy
NAME          POD-SELECTOR   AGE
allow-a-to-b  app=backend    21s
deny-to-c     app=blocked    13s
```

```
abhai@abhai-VirtualBox:~$ kubectl get pods
NAME                       READY   STATUS    RESTARTS      AGE
backend-7f7658c6bc-gxhn2   1/1     Running   4 (54m ago)   12d
blocked-765c449c7f-zjv7m   1/1     Running   0             3m43s
web-65d846d465-277w8       1/1     Running   4 (54m ago)   12d
```

Step 3: Apply Microsegmentation Policies

Create and apply network policies to enforce traffic control between pods:

- **Allow_A_to_B.yaml** — Allows the backend pod to communicate with the web pod.
- **Deny_to_C.yaml** — Blocks specific pod communication.

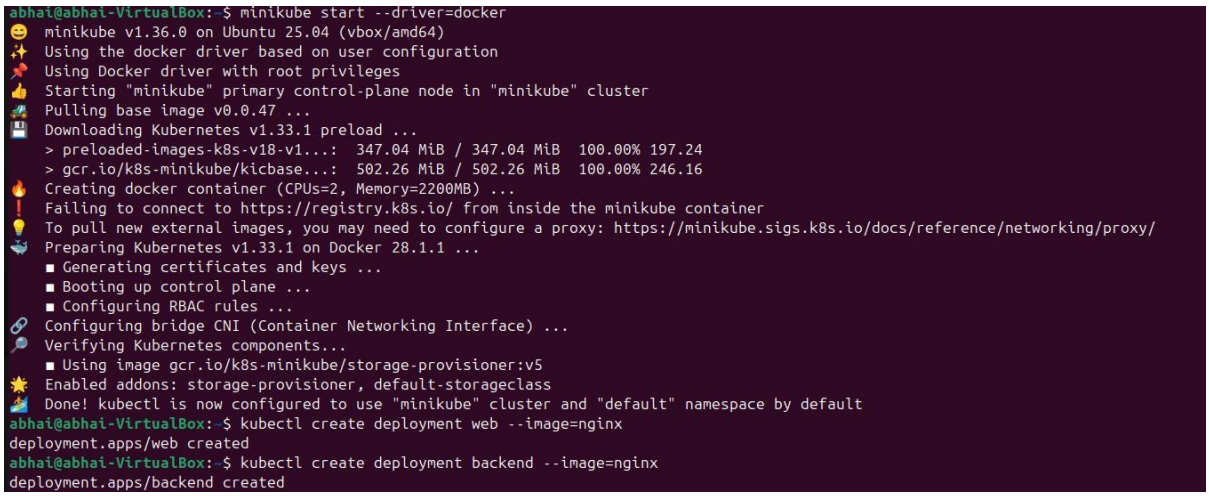
Apply both policies using kubectl apply. Verify the policies are active with kubectl get networkpolicy.

Step 4: Validate Pod Labels and Connectivity

Check pod labels using kubectl get pods --show-labels. These labels (app=backend, app=web) are essential for network policy selectors.

- Test allowed connections to confirm that the **backend → web** path works.
- Test denied paths to ensure that blocked traffic is enforced as expected.

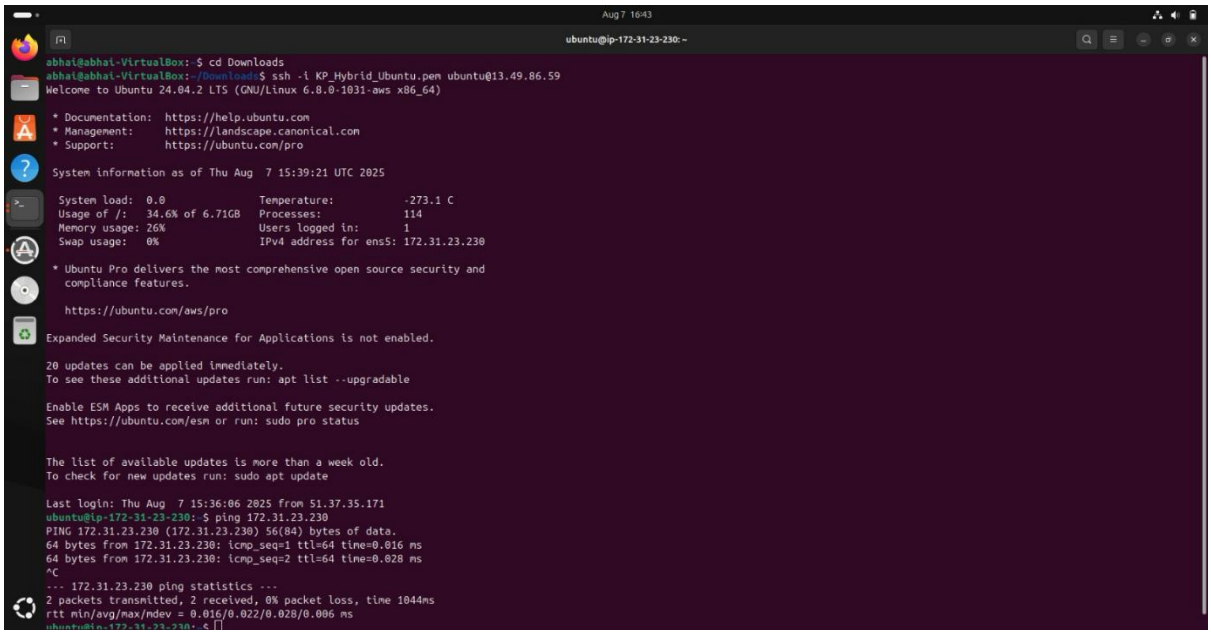
Outcome:
This setup demonstrates **Calico microsegmentation**, where specific pod-to-pod communication is explicitly allowed or denied, improving cluster security and reducing the risk of lateral movement.



Ping and Connectivity Testing

1. Establishing SSH Access

The target system is accessed over SSH, confirming system readiness (memory, CPU, IP address). A ping test to the Kubernetes pod/service IP validates connectivity at the network level.



2. Executing Ping Inside Pods

From the local Kubernetes environment, kubectl exec is used to enter the backend pod shell. From within the pod, ping commands to other pods/services confirm whether the Calico network policies are correctly allowing or denying connections.

Expected Results:

- Pings from allowed pods succeed.
- Pings from denied pods fail, proving the network policies are working as intended.

```
abhai@abhai-VirtualBox:~$ kubectl get pods
NAME                       READY   STATUS    RESTARTS      AGE
backend-7f7658c6bc-gxhn2   1/1     Running   1 (19m ago)   8d
web-65d846d465-277w8       1/1     Running   1 (19m ago)   8d
abhai@abhai-VirtualBox:~$ kubectl exec -it backend-7f7658c6bc-gxhn2 -- /bin/sh

#
# 
```

3. VPN vs Non-VPN Latency

Latency testing was conducted to measure the impact of VPN:

- **Without VPN** → Average round-trip time ~233**ms**
- **With VPN** → Average round-trip time ~ **350ms**

The results confirmed that VPN tunneling increased latency due to encryption overhead and longer routing paths.

Latency Without VPN

```
abhai@abhai-VirtualBox:~$ ping -c 10 13.49.86.59
PING 13.49.86.59 (13.49.86.59) 56(84) bytes of data.
64 bytes from 13.49.86.59: icmp_seq=1 ttl=255 time=50.0 ms
64 bytes from 13.49.86.59: icmp_seq=2 ttl=255 time=92.2 ms
64 bytes from 13.49.86.59: icmp_seq=3 ttl=255 time=131 ms
64 bytes from 13.49.86.59: icmp_seq=4 ttl=255 time=174 ms
64 bytes from 13.49.86.59: icmp_seq=5 ttl=255 time=212 ms
64 bytes from 13.49.86.59: icmp_seq=6 ttl=255 time=257 ms
64 bytes from 13.49.86.59: icmp_seq=7 ttl=255 time=291 ms
64 bytes from 13.49.86.59: icmp_seq=8 ttl=255 time=331 ms
64 bytes from 13.49.86.59: icmp_seq=9 ttl=255 time=375 ms
64 bytes from 13.49.86.59: icmp_seq=10 ttl=255 time=418 ms

--- 13.49.86.59 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9020ms
rtt min/avg/max/mdev = 50.041/233.054/417.925/116.508 ms
abhai@abhai-VirtualBox:~$
```

Latency with VPN

```
abhai@abhai-VirtualBox:~$ ping -c 10 172.31.23.230
PING 172.31.23.230 (172.31.23.230) 56(84) bytes of data.
64 bytes from 172.31.23.230: icmp_seq=1 ttl=64 time=189 ms
64 bytes from 172.31.23.230: icmp_seq=2 ttl=64 time=225 ms
64 bytes from 172.31.23.230: icmp_seq=3 ttl=64 time=260 ms
64 bytes from 172.31.23.230: icmp_seq=4 ttl=64 time=296 ms
64 bytes from 172.31.23.230: icmp_seq=5 ttl=64 time=331 ms
64 bytes from 172.31.23.230: icmp_seq=6 ttl=64 time=366 ms
64 bytes from 172.31.23.230: icmp_seq=7 ttl=64 time=402 ms
64 bytes from 172.31.23.230: icmp_seq=8 ttl=64 time=439 ms
64 bytes from 172.31.23.230: icmp_seq=9 ttl=64 time=477 ms
64 bytes from 172.31.23.230: icmp_seq=10 ttl=64 time=515 ms

--- 172.31.23.230 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9032ms
rtt min/avg/max/mdev = 188.948/349.861/514.515/103.331 ms
```