National College of Ireland

Configuration Manual

MSc Research Project
MSc. Cyber Security

Sivadas Mundollikalam
Student ID: 23408219

School of Computing
National College of Ireland

Supervisor: Mr. Michael Prior

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sivadas Mundollikalam		
Student ID:	23408219		
Programme:	MSc. Cyber Security	**Year:**	2025
Module:	MSc (Research) Practicum		
Supervisor:	Michael Prior		
Submission Due Date:	11-12-2025		
Project Title:			
Word Count:	**842**	**Page Count:7**	

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: 

Date: 10-12-2025

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

Sivadas Mundollikalam

23408219

1. Introduction

Modern e-commerce platforms are increasingly facing automated bot traffic, malicious scraping, and high-frequency DDoS-style request bursts that traditional static rate-limiters can no longer handle. Fixed global thresholds fail to distinguish between the normal user activities, legitimate traffic spikes and genuine attacks leading either to unnecessary throttling of real customers or delayed detection of malicious clients. To address this gap, this project implements a Redis–Lua–based Adaptive Rate-Limiting System integrated into a FastAPI e-commerce backend. The system dynamically adjusts per IP request thresholds in real time, enabling accurate identification of anomalies such as the bot surges, brute-force attempts and scraping behaviour. Unlike the static rate-limiters and the adaptive model responds to traffic patterns by increasing or tightening limits automatically, improving resilience without degrading user experience. This configuration manual provides the complete installation, deployment, and operational steps required to set up and run the adaptive rate-limiting environment on both local and AWS infrastructures.

FastAPI backend running core e-commerce logic

- **Redis + Lua atomic scripts** implementing the adaptive limiter
- **Prometheus metrics exporter** collecting real-time telemetry
- **Grafana dashboards** for visualization of hits, IP patterns, 429 blocks, and system health
- **AWS EC2 deployment instructions** for production-ready cloud execution

2. System Requirements

2.1 Software Versions

Component	Version
Python	**3.12.3**
FastAPI	**0.100.0**
Redis Server	**7.2.15**
redis-py client	**5.0.1**
Uvicorn	**0.22.0**
Prometheus	**2.49.0**
Grafana	**12.3.0**
Jinja2	**3.1.2**
prometheus-client	**0.17.0**
aioredis	**2.0.1**

2.2 Hardware Requirements

Local:

- CPU: 2+ cores
- RAM: 4GB
- Disk: 4GB free

AWS EC2:

- Instance type: **t2.micro / t3.micro**
- OS: **Ubuntu 22.04 LTS**

3. Project Directory Structure

/ecommerce-adaptive-rate/

```
│
├── app/
│   ├── main.py              # FastAPI application
│   ├── limiter.py           # Static/adaptive Lua-based middleware
│   ├── metrics.py           # Prometheus counters & histograms
│   ├── templates/
│   │   └── cartpage.html
│   └── __init__.py
│
├── requirements.txt
└── README.md
```

4. Installation and Local Setup

4.1 Create Virtual Environment

sudo apt update

sudo apt install python3-venv redis-server -y

python3 -m venv venv

source venv/bin/activate

4.2 Install Python Dependencies

pip install -r requirements.txt

4.3 Start Redis

sudo systemctl enable redis-server

sudo systemctl start redis-server

4.4 Run Application Locally

uvicorn app.main:app --host 0.0.0.0 --port 8000

Local test:

curl http://127.0.0.1:8000

curl http://127.0.0.1:8000/metrics

Load test:

```
for i in {1..20}; do
  curl -s -o /dev/null -w "%{http_code}\n" -X POST http://127.0.0.1:8000/add_to_cart;
done
```

5. AWS EC2 Deployment (Production Setup)

5.1 Launch EC2

- Login to your AWS console → EC2 → Launch instance
- Choose Ubuntu 22.04 LTS
- Instance type: t2.micro (Free Tier) or higher if testing heavy traffic
- Create a new key pair (.pem) and download it
- Security group settings → Allow:

Port	Purpose
22	SSH
80	HTTP
8000	Testing
9090	Prometheus
3000	Grafana

5.2 Install Dependencies

sudo apt update

sudo apt install python3-venv redis-server nginx -y

5.3 Copy Project to EC2

git clone https://github.com/SivNCI/Ecom.git

5.4 Create Unicorn Service

CREATE /ETC/SYSTEMD/SYSTEM/FASTAPI.SERVICE:

==========

```
sudo tee /etc/systemd/system/fastapi.service > /dev/null <<'EOF'
[Unit]
Description=FastAPI adaptive rate limiter
After=network.target
```

[Service]

User=ubuntu

Group=ubuntu

WorkingDirectory=/home/ubuntu/ecommerce-adaptive-rate

Environment="PATH=/home/ubuntu/ecommerce-adaptive-rate/venv/bin"

ExecStart=/home/ubuntu/ecommerce-adaptive-rate/venv/bin/uvicorn app.main:app --host 0.0.0.0 --port 8000 --workers 1

Restart=always

RestartSec=3

[Install]

WantedBy=multi-user.target

EOF

==========

sudo systemctl daemon-reload

sudo systemctl start fastapi.service

sudo systemctl enable fastapi.service

sudo journalctl -u fastapi.service -f

6. Redis + Lua Configuration

Adaptive Lua Script Logic

Located inside **limiter.py**.

- Calculates request rate
- Detects surges
- Adjusts limit dynamically
- Returns:
    - Remaining quota
    - "BLOCK" when limit exceeded

7. Prometheus Installation & Setup

7.1 Install Prometheus

wget https://github.com/prometheus/prometheus/releases/latest/download/prometheus-*.linux-amd64.tar.gz

tar -xvf prometheus*.tar.gz

sudo mv prometheus-*/ /etc/prometheus/

sudo useradd --no-create-home prometheus

7.2 Prometheus Config

/etc/prometheus/prometheus.yml

```
global:
  scrape_interval: 5s
scrape_configs:
  - job_name: "fastapi"
    static_configs:
      - targets: ["localhost:8000"]
```

Start service:

```
sudo systemctl restart prometheus
sudo systemctl enable prometheus
```

8. Grafana Installation & Setup

8.1 Install Grafana

```
sudo apt install -y apt-transport-https software-properties-common
sudo apt install grafana -y
sudo systemctl enable grafana-server
sudo systemctl start grafana-server
```

Open browser:

http://<EC2 Public IP>:3000

User name: admin

Password: Siva

9. Grafana Dashboards

Custom Dashboard Panels:

- Top IPs
- Per-IP request counts
- 429 status count
- Latency graph
- Traffic distribution
- Adaptive limit behaviour

Required Queries

9.1 Total hits per IP

```
sum(request_count_total{
   ip!="172.31.68.236",
   endpoint!="/metrics" })
```

Note: Excluding EC2 private ip:172.31.68.236 for avoiding log metrics hits

9.2 Count of 429 Requests

```
sum(request_status_total{status="429"})
```

9.3 Total POST response hit

```
sum(request_count_total{
   method="POST"})
```

9.4 IP-wise Traffic Table

```
sum by (ip, method) (
  request_count_total{
   ip!="172.31.68.236",
   endpoint!="/metrics"})
```

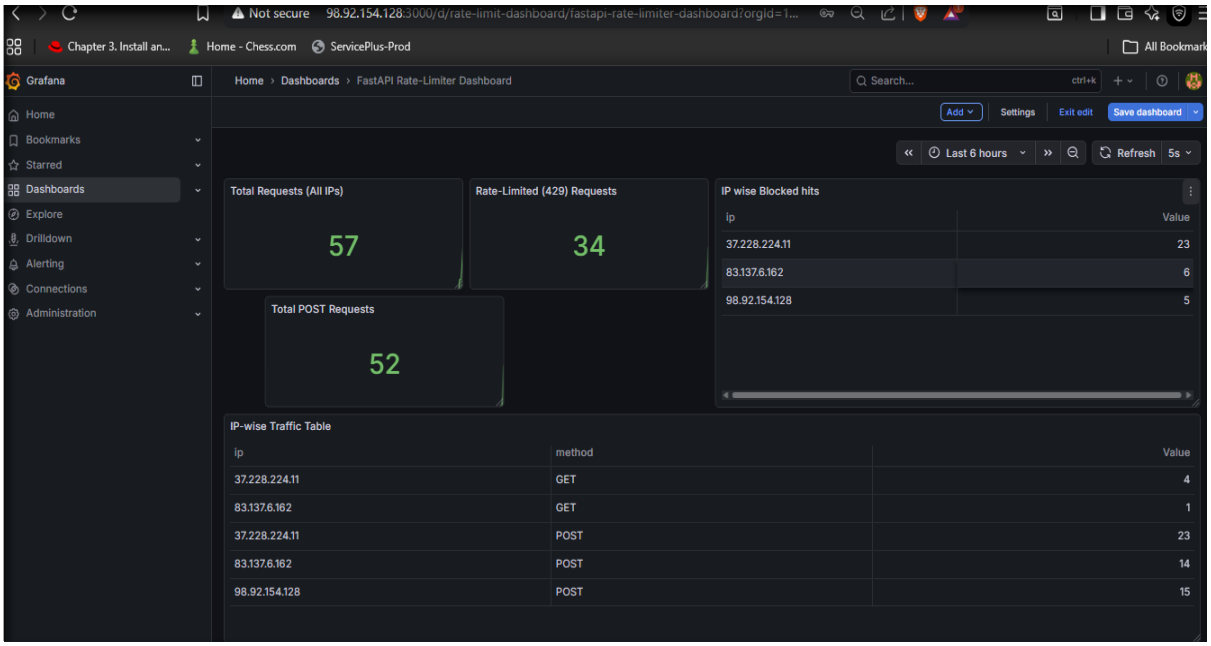

Fig:1

10. Configuration Parameters

10.1 Adaptive Rate Limit Parameters

Located in **limiter.py**

Parameter	Purpose
base_limit	Normal maximum requests
surge_threshold	When traffic spikes
penalty_limit	Block threshold
decay_window	Recovery time

10.2 Prometheus Metrics

Stored in **metrics.py**

Metric	Description
request_count_total	Track all requests
request_status_total	Track 200, 429
request_latency_seconds	Histogram
adaptive_limit_current	Current dynamic limit

11. Testing

for i in {1..20}; do curl -s -o /dev/null -w "%{http_code}\n" -X POST http:// <*EC2 Public IP*>:8000/add_to_cart; done

12. Troubleshooting

Redis Evalsha error

Cause: wrong parameters format
Fix used:

redis.evalsha(sha, 1, key, arg1, arg2)

Prometheus "connection refused"

Prometheus not running
Firewall issue
FastAPI not exposed

Grafana "no data"

Range too small
Wrong datasource UID
Service slow due to traffic

13. Conclusion

This manual provides the complete configuration required to deploy, monitor, and maintain the Adaptive Rate-Limiting system in both local and cloud environments. All modules are modular and extensible, supporting production deployment and further academic research.