

# **Cloud Computing**

## **LAB-2**

Name: Vardha Kathuria

Section: F

SRN:PES1UG23AM343

## Events

Welcome PES1UG23AM343. Register for events below.

[View My Events →](#)

Event ID: 1

₹ 500

### Hackathon

Includes certificate • instant registration • limited seats

[Register](#)

Event ID: 2

₹ 300

### Dance

Includes certificate • instant registration • limited seats

[Register](#)

Event ID: 3

₹ 500

### Hackathon

Includes certificate • instant registration • limited seats

[Register](#)

Event ID: 4

₹ 300

### Dance Battle

Includes certificate • instant registration • limited seats

[Register](#)

Event ID: 5

₹ 400

### AI Workshop

Includes certificate • instant registration • limited seats

[Register](#)

Event ID: 6

₹ 200

### Photography Walk

Includes certificate • instant registration • limited seats

[Register](#)

Event ID: 7

₹ 350

Event ID: 8

₹ 250

Event ID: 9

₹ 150

```
● INFO: Started reloader process [18460] using StatReload
INFO: Started server process [7808]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:6539 - "GET /register HTTP/1.1" 200 OK
INFO: 127.0.0.1:6539 - "GET /favicon.ico HTTP/1.1" 404 Not Found
INFO: 127.0.0.1:44208 - "POST /register HTTP/1.1" 302 Found
INFO: 127.0.0.1:44208 - "GET /login HTTP/1.1" 200 OK
INFO: 127.0.0.1:33026 - "POST /login HTTP/1.1" 302 Found
INFO: 127.0.0.1:33026 - "GET /events?user=PES1UG23AM343 HTTP/1.1" 200 OK
INFO: 127.0.0.1:52665 - "GET /checkout HTTP/1.1" 500 Internal Server Error
```

# Route 1: checkout

**Fest Monolith**  
FastAPI • SQLite • Locust

**Login** **Create Account**

## Monolith Failure

HTTP 500

One bug in one module impacted the **entire application**.

**Error Message**  
division by zero

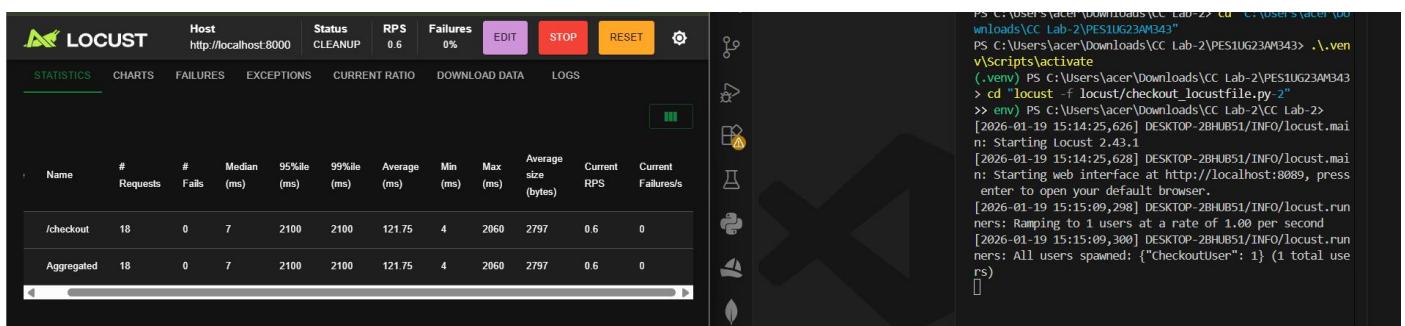
**Why did this happen?**  
Because this is a **monolithic application**: all modules share the same runtime and deployment. When one feature crashes, it affects the whole system.

**What should you do in the lab?**

- Take a screenshot (crash demonstration)
- Fix the bug in the indicated module
- Restart the server and verify recovery

**Back to Events** **Login**

## Before optimization:





## Checkout

This route is used to demonstrate a monolith crash + optimization.

Total Payable

**₹ 9500**

After fixing + optimizing checkout logic, re-run Locust and compare results.

### What you should observe

- One buggy feature can crash the entire monolith.
- Inefficient loops cause high response times under load.
- Optimization improves performance but architecture still scales as one unit.

Next Lab: Split this monolith into Microservices (Events / Registration / Checkout).

### After optimization:

The screenshot shows the Locust interface running on <http://localhost:8000>. The statistics table indicates 18 requests, 0 fails, and a median response time of 2100ms. The terminal window to the right shows the command-line session where the user navigated to the project directory, activated the virtual environment, and ran the Locust command. The output shows the Locust web interface starting at <http://localhost:8089>.

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures	
GET	/checkout	18	0	6	2100	2100	121.12	3	2065	2797	0.7	0
Aggregated												
18 0 6 2100 2100 121.12 3 2065 2797 0.7 0												

```

3 def checkout_logic
4     db = get_db()
5     db.row_factory
6     events = db.ex
7     # 1 / 0
8
9     total = 0
10    for e in event:
11        fee = e[0]
12        for e in e:
13            total =
14        return tot
15
16
17
18
19
  \.venv\Scripts\activate' is not recognized
as the name of a cmdlet, function, script
file, or operable program. Check the spelling
of the name, or if a path was included,
verify that the path is correct and try again.
At line:1 char:1
+ \.venv\Scripts\activate
+ ~~~~~
+ CategoryInfo          : ObjectNotFound:
(.\venv\Scripts\activate:String) [], Co
mmandNotFoundException
+ FullyQualifiedErrorId : CommandNotFound
Exception
  PS C:\Users\acer\Downloads\CC Lab-2> cd "C:\Users\acer\Downloads\CC Lab-2\PES1UG23AM343"
  PS C:\Users\acer\Downloads\CC Lab-2\PES1UG23AM343 > \.ven
  v\Scripts\activate
  (.\venv) PS C:\Users\acer\Downloads\CC Lab-2\PES1UG23AM343
  > cd "C:\Users\acer\Downloads\CC Lab-2\PES1UG23AM343"
  PS C:\Users\acer\Downloads\CC Lab-2\PES1UG23AM343 > locust
  -f locust/checkout_locustfile.py
  [2026-01-19 15:20:41,364] DESKTOP-2BHUB51/INFO/locust.mai
n: Starting Locust 2.43.1
  [2026-01-19 15:20:41,365] DESKTOP-2BHUB51/INFO/locust.mai
n: Starting web interface at http://localhost:8089, press
  enter to open your default browser.
  [2026-01-19 15:21:13,952] DESKTOP-2BHUB51/INFO/locust.run
ers: Ramping to 1 users at a rate of 1.00 per second
  [2026-01-19 15:21:13,953] DESKTOP-2BHUB51/INFO/locust.run
ers: All users spawned: {"checkoutUser": 1} (1 total use
rs)
  
```

## **Bottleneck**

The /checkout route initially contained an intentional bug (division by zero) which caused a server crash. Additionally, the fee calculation logic was inefficient, as it used a loop to increment the total amount one unit at a time.

## **Change Made**

- The division-by-zero statement was commented out to prevent the application from crashing.
- The fee calculation logic was optimized by directly summing the event fees instead of using repetitive looping.

## **Why Performance Improved**

Removing the crash ensured application stability, and optimizing the fee calculation reduced unnecessary CPU computation. This resulted in a lower average response time during load testing.

# Route 2: events

## Before optimization:

The screenshot shows the Locust interface with the following details:

- Host:** http://localhost:8000
- Status:** RUNNING
- Users:** 1
- RPS:** 0.44
- Failures:** 0%
- STATISTICS:** Shows a table with one row for the GET request to '/events?user=locust\_user'.
- CHARTS:** Shows a chart with data points for Median, 95%ile, and 99%ile response times.
- EXCEPTIONS:** Shows a table with one row for the same GET request.
- CURRENT RATIO:** Shows a ratio of 1.
- DOWNLOAD DATA:** Shows a table with one row for the same GET request.
- LOGS:** Shows command-line logs for Locust setup and start.
- Code Editor:** Displays Python code for a user task named 'EventsUser'. It includes imports for 'db', 'events', and 'fee'. The task logic involves a loop over events, calculating fees, and summing them up.

## After optimization:

The screenshot shows the Locust interface after optimization, with the following changes:

- Host:** http://localhost:8000
- Status:** RUNNING
- Users:** 1
- RPS:** 0.6
- Failures:** 0%
- STATISTICS:** Shows a table with one row for the same GET request, showing improved performance metrics.
- CHARTS:** Shows a chart with data points for Median, 95%ile, and 99%ile response times.
- EXCEPTIONS:** Shows a table with one row for the same GET request.
- CURRENT RATIO:** Shows a ratio of 1.
- DOWNLOAD DATA:** Shows a table with one row for the same GET request.
- LOGS:** Shows command-line logs for Locust setup and start.
- Code Editor:** Displays the same Python code for the 'EventsUser' task, which appears to be identical to the original version.

## **Bottleneck**

The /events route contained an unnecessary computation loop that executed millions of iterations without contributing to the actual functionality. This caused artificial CPU delay and increased response time for every request.

## **Change Made**

The redundant loop was completely removed, leaving only the required database query and template rendering logic.

## **Why Performance Improved**

By eliminating unnecessary CPU-intensive operations, the server was able to respond faster to requests. This significantly reduced the average response time observed in Locust after optimization.

# Route 3: myevents

Before optimization:

The screenshot shows the Locust interface with two tabs: 'STATISTICS' and 'LOGS'. The 'STATISTICS' tab displays performance metrics for a single request type: GET /my-events?user=locust\_user. The 'LOGS' tab shows the source code for the application, which includes routes for '/register\_event', '/my-events', and '/checkout', along with database operations using SQLAlchemy.

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s	
GET	/my-events?user=locust_user	7	0	77	2100	2100	372.37	75	2135	3144	0.5	0
	Aggregated	7	0	77	2100	2100	372.37	75	2135	3144	0.5	0

```
(.venv) PS C:\Users\locust\locust\myevents_locustfile.py
[2026-01-19 15:31:23,533] DESKTOP-2BHUB51/INFO/locust.main: Starting Locust 2.43.1
[2026-01-19 15:31:23,534] DESKTOP-2BHUB51/INFO/locust.main: Starting web interface at http://localhost:8089, press enter to open your default browser.
[2026-01-19 15:31:37,422] DESKTOP-2BHUB51/INFO/locust.runners: Ramping to 1 users at a rate of 1.00 per second
[2026-01-19 15:31:37,423] DESKTOP-2BHUB51/INFO/locust.runners: All users spawned: {"MyEventsUser": 1} (1 total users)
```

After optimization:

The screenshot shows the Locust interface with the same tabs and data as the previous version. The 'STATISTICS' tab now shows improved performance metrics for the same request type. The 'LOGS' tab shows the optimized source code, which includes a more efficient database query for retrieving events.

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s	
GET	/my-events?user=locust_user	6	0	6	2100	2100	353.44	3	2094	3144	0.43	0
	Aggregated	6	0	6	2100	2100	353.44	3	2094	3144	0.43	0

```
68
69
70
71 @app.get("/register_event")
72 def register_event(event):
73     if event_id == 404:
74         1 / 0
75
76     db = get_db()
77     db.execute("INSERT INTO events...")
78     db.commit()
79
80     return RedirectResponse(
81         "/",
82     )
83
84 @app.get("/my-events")
85 def my_events(request):
86     db = get_db()
87
88     rows = db.execute(
89         """
90         SELECT name, fe
91         FROM events
92         WHERE id IN (
93             SELECT ever
94             FROM regist
95             WHERE user_
96             ...
97             (user,)
98         ).fetchall()
99
100    return templates.TemplateResponse(
101        "my_events.html",
102        {"request": req
103    }
104
105 @app.get("/checkout")
106 def checkout(request):
107     total = checkout.lc
108     return templates.TemplateResponse(...)
```

## **Bottleneck**

The /my-events route had an artificial delay introduced through a large loop performing over a million iterations. Additionally, the original JOIN-based query added extra processing overhead.

## **Changes Made**

- The artificial computation loop was removed.
- The database query was simplified by using a subquery instead of a JOIN to fetch only the required event details.

## **Why Performance Improved**

Removing the redundant computation eliminated CPU wastage, and simplifying the database query reduced execution overhead. Together, these changes improved response time and overall performance under load.

## **Conclusion**

This lab demonstrated the limitations of monolithic architecture, where inefficiencies or errors in a single module can impact the entire application. By identifying bottlenecks and optimizing individual routes, measurable performance improvements were observed using Locust. This highlights the importance of efficient backend logic and serves as a motivation for adopting more modular architectures such as microservices.

---