

Cloud Computing - Lab 2

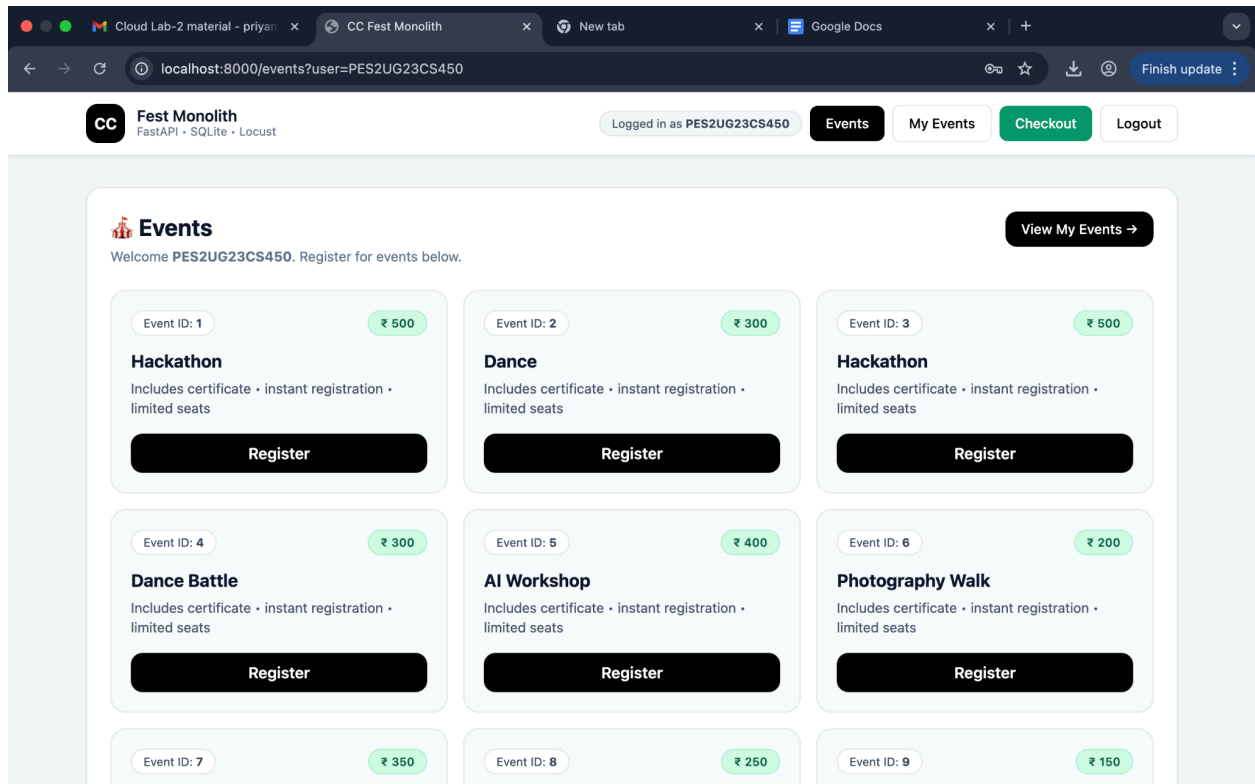
Monolithic Architecture

PES2UG23CS450
Priyanka Satish

SEM VI
Section G

SS1

Events page after logging in



SS2

Crash while check out (uncommented line of code)

Cloud Lab-2 material - priya x CC Fest Monolith x New tab x Untitled document - Google x +

localhost:8000/register_event/404?user=PES2UG23CS450

Finish update

CC Fest Monolith
FastAPI • SQLite • Locust

Logged in as PES2UG23CS450 Events My Events Checkout Logout

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

CC Week X • Monolithic Applications Lab

```
INFO: 127.0.0.1:56531 - "GET /checkout HTTP/1.1" 500 Internal Server Error
ERROR: Exception in ASGI application
```

SS3

No crash (commented line of code)

The screenshot displays a web browser window with the address bar showing `localhost:8000/checkout`. The page header includes the 'Fest Monolith' logo, the text 'FastAPI • SQLite • Locust', and buttons for 'Login' and 'Create Account'. The main content area is divided into two columns. The left column, titled 'Checkout', contains a message about demonstrating a monolith crash and optimization, a 'Total Payable' of ₹ 6600, and a green box with a checkmark indicating that after fixing and optimizing the checkout logic, the user should re-run Locust and compare results. The right column, titled 'What you should observe', lists three bullet points: 'One buggy feature can crash the entire monolith.', 'Inefficient loops cause high response times under load.', and 'Optimization improves performance but architecture still scales as one unit.' Below this is a yellow box with the text 'Next Lab: Split this monolith into Microservices (Events / Registration / Checkout)'. At the bottom left, there is a footer that reads 'CC Week X • Monolithic Applications Lab'. Below the main content area, a terminal window shows two log entries: `INFO: 127.0.0.1:56563 - "GET /events?user=PES2UG23CS450 HTTP/1.1" 200 OK` and `INFO: 127.0.0.1:56563 - "GET /checkout HTTP/1.1" 200 OK`.

CC Fest Monolith
FastAPI • SQLite • Locust

Login Create Account

Checkout

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

Total Payable
₹ 6600

✓ 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.

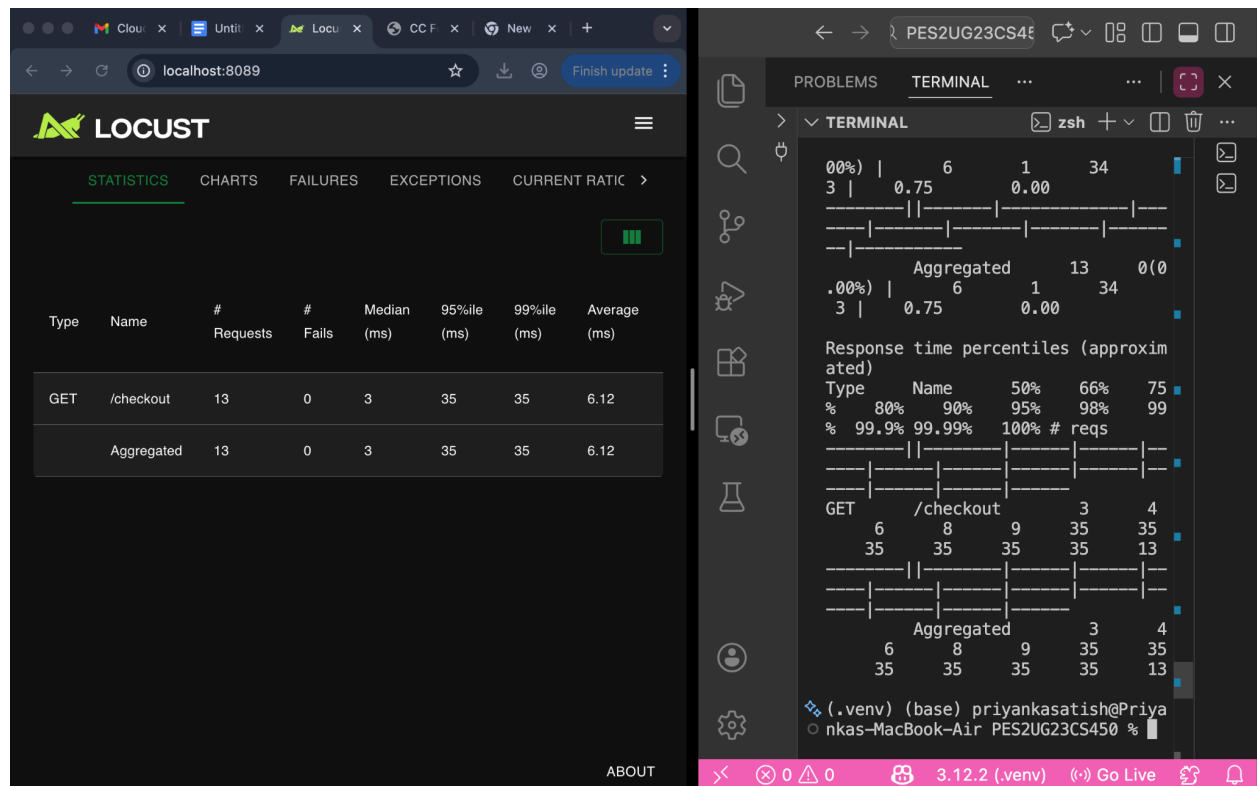
Next Lab: Split this monolith into Microservices (Events / Registration / Checkout).

CC Week X • Monolithic Applications Lab

```
INFO: 127.0.0.1:56563 - "GET /events?user=PES2UG23CS450 HTTP/1.1" 200 OK
INFO: 127.0.0.1:56563 - "GET /checkout HTTP/1.1" 200 OK
```

SS4

Locust before code optimization for checkout (avg ms = 6.12)

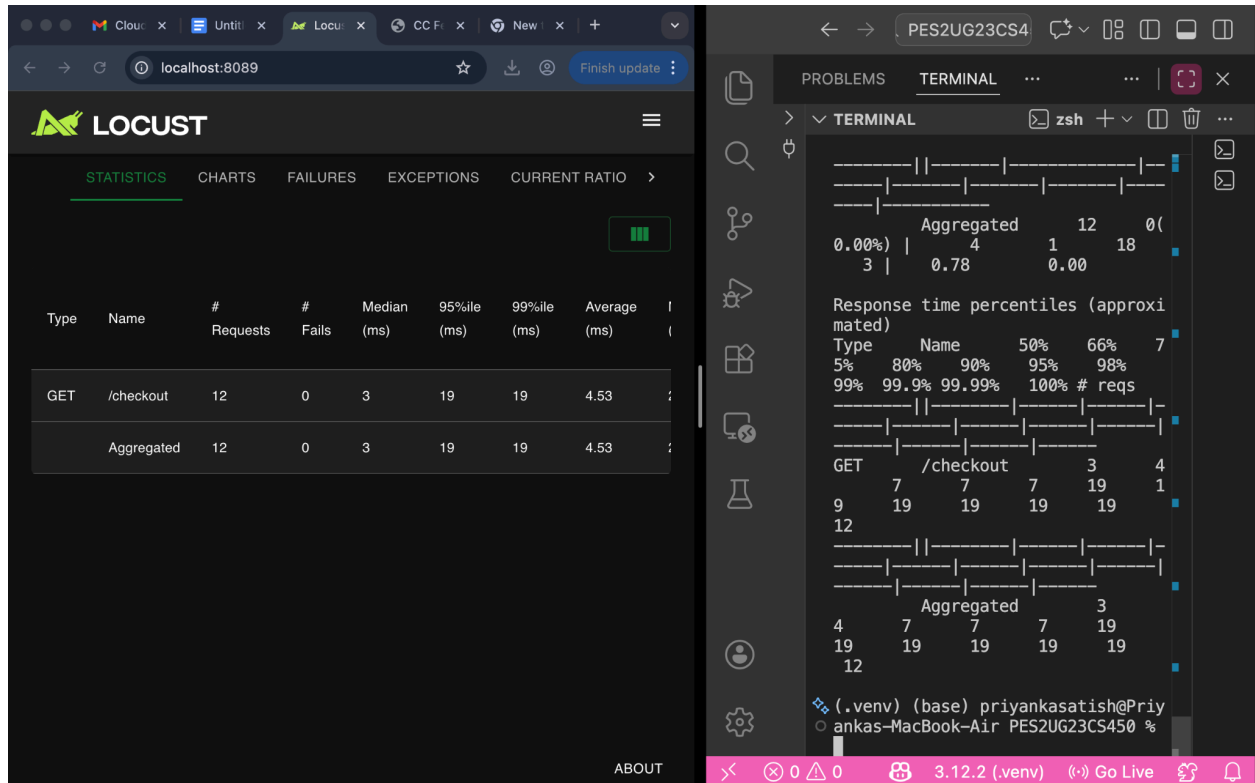


Inefficient code :

```
total = 0
for e in events:
    fee = e[0]
    while fee > 0:
        total += 1
        fee -= 1
```

SS5

Locust after code optimization for checkout avg ms = (4.53)



Efficient code :

```
total = 0
for e in events:
    total += e[0]
```

SS6

Locust before code optimization for events (avg ms = 121.63)

The screenshot displays the Locust web interface at localhost:8089 and a terminal window. The web interface shows the 'STATISTICS' tab with a table of performance metrics for a GET request to /events?user=locust_user. The table indicates 13 requests, 0 fails, a median response time of 120ms, and an average response time of 121.63ms. The terminal window shows the command 'locust -f locust/events_locustf.py' being executed, followed by the output of the 'Response time percentiles' command, which displays a detailed breakdown of response times for the same endpoint.

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)
GET	/events?user=locust_user	13	0	120	150	150	121.63
Aggregated		13	0	120	150	150	121.63

```
(.venv) (base) priyankasatish@PES2UG23CS450
ankas-MacBook-Air PES2UG23CS450
locust -f locust/events_locustf.py

Response time percentiles (approximated)
Type      Name      50%    66%
75%      80%    90%    95%    98%
99%    99.9% 99.99% 100% # reqs
-----|-----|-----|-----|
GET      /events?user=locust_user
120     130     130     130
140     150     150     150     130
150     150     13
-----|-----|-----|-----|
Aggregated      120     1
30     130     130     140     150
150     150     150     150     150
13
```

SS7

Locust after code optimization for events (avg ms = 9.09)

The screenshot displays the Locust web interface at localhost:8089 and a terminal window. The interface shows statistics for a GET request to /events?user=locust_user, with an average response time of 9.09 ms. The terminal shows the command locust -f locust/events_locustfile being executed, resulting in a response time of 0.70 ms.

Locust Web Interface Statistics:

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)
GET	/events?user=locust_user	13	0	4	60	60	9.09
Aggregated		13	0	4	60	60	9.09

Terminal Output:

```
(.venv) (base) priyankasatish@Pri  
(.venv) (base) priyankasatish@Pri  
ankas-MacBook-Air PES2UG23CS450 %  
locust -f locust/events_locustfile  
...  
4 | 0.70 0.00  
Response time percentiles (approx  
imated)  
Type Name 50% 66% 7  
5% 80% 90% 95% 98%  
99% 99.9% 99.99% 100% # reqs  
-----|-----|-----|-----|  
-----|-----|-----|-----|  
GET /events?user=locust_user  
4 5 7 9  
17 60 60 60 60  
60 60 13  
-----|-----|-----|-----|  
-----|-----|-----|-----|  
Aggregated 4  
5 7 9 17 60  
60 60 60 60 60  
13
```

Optimization :

```
waste = 0  
for i in range(3000000):  
    waste += i % 3
```

Unnecessary loop removed

SS8

Locust before optimization for myevents (avg ms = 63.18)

The screenshot displays the Locust web interface on the left and a terminal window on the right. The Locust interface shows the 'STATISTICS' tab with a table of performance metrics for a GET request to `/my-events?user=locust_user`. The table indicates 11 requests, 0 fails, a median response time of 71ms, and an average response time of 63.18ms. The terminal window shows the output of the Locust CLI, including response time percentiles and a detailed breakdown of the request metrics.

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)
GET	/my-events?user=locust_user	11	0	71	82	82	63.18
Aggregated		11	0	71	82	82	63.18

```
-----|-----|-----|-----|
-----|-----|-----|-----|
-----|-----|-----|-----|
Aggregated      11      0(
0.00%) |      63      39      81
71 |      0.68      0.00

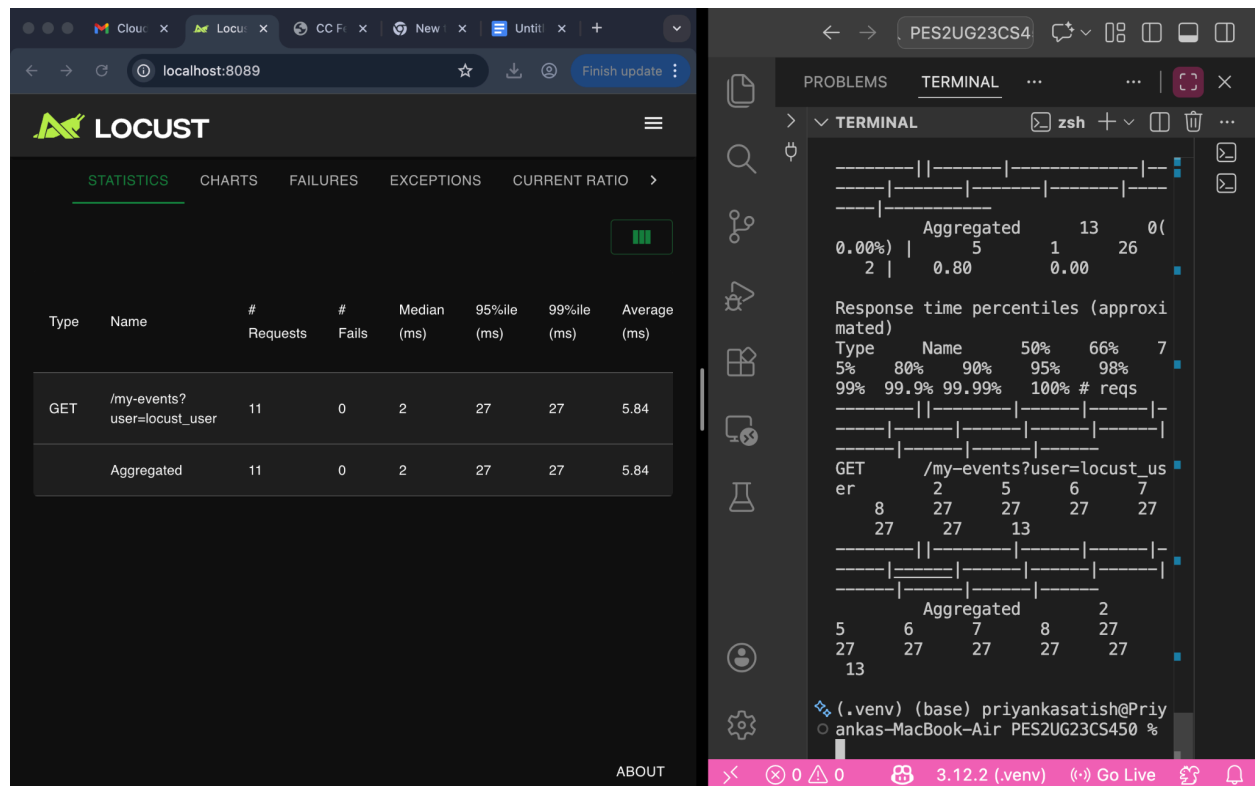
Response time percentiles (approx
imated)
Type      Name      50%      66%      7
5%      80%      90%      95%      98%
99%      99.9%      99.99%      100% # reqs
-----|-----|-----|-----|
-----|-----|-----|-----|
GET      /my-events?user=locust_us
er      71      73      74      74
75      82      82      82      82
82      82      11

-----|-----|-----|-----|
-----|-----|-----|-----|
-----|-----|-----|-----|
Aggregated      71      7
3      74      82      82
3      74      74      75      82
82      82      82      82      82
11
```

Terminal output (bottom):
❖ (.venv) (base) priyankasatish@Priy
ankas-MacBook-Air PES2UG23CS450 %

SS9

Locust after optimization for myevents (avg ms = 5.84)



Optimization:

```
dummy = 0
for _ in range(1500000):
    dummy += 1
```

Unnecessary loop removed

GitHub link : <https://github.com/PES2UG23CS450/CC-LAB2-Monolithic-Architecture.git>