Swarm Savvy: Transform Mainframe Testing with Open Source Tools

Adam Munawar Rahman

Staff Software Developer @ IBM

Medium: @msradam

LinkedIn: /in/adamsrahman

Marist Computing Conference, November 2025

Load Testing Failures Cost Millions

Three high-profile failures demonstrate why load testing matters:

- 🎫 **Ticketmaster (Nov 2022)** - Taylor Swift presale: 3.5B requests (4x capacity). Site crashed. Multiple lawsuits filed. [1]
- 🎮 **Nintendo Switch 2 (Apr 2025)** - Multiple retailers crashed. 2.2M+ applications in Japan alone. [2]
- 🏛️ **TSB Bank (Apr 2018)** - £330M impact, 5.2M customers locked out for weeks. [3]

Bottom line: Load testing reveals limits before customers feel them.

[1] "Taylor Swift–Ticketmaster controversy," Wikipedia, 2025
[2] "Nintendo Switch 2 Pre-Order Demand Outpaces Expectations," Game Informer, Apr. 2025
[3] "TSB Bank Data Migration Failure," iceDQ, Jan. 2025

Mainframes Handle Critical Workloads

Mainframes process 1M+ transactions per second:

- **90% of credit card transactions** worldwide [4]
- **92 of top 100 banks** rely on mainframe systems [5]
- Testing practices haven't kept pace with modern DevOps

The opportunity: Modern open-source tools for mainframe testing.

[4] "9 Mainframe Statistics That May Surprise You," Precisely, Sep. 2024
[5] "How Do Banks Maintain Financial Data?" Bank Systems & Technology

The Gap: Legacy vs Modern Tools

Legacy tools are resource-heavy:

- IBM TPNS (1976), Apache JMeter (1998), IBM WSim (2002)
- Thread-per-user: **1MB+ per thread** [6,7]
- GUI-driven, XML configuration files [8,9]

Modern tools are efficient:

- Locust (2011), Grafana k6 (2017)
- Event-driven: **14-30x less memory, 100K+ users** [6,7]
- Python/JavaScript, CLI-native, pipeline-ready [8,9]

[6] N. van der Hoeven, "Comparing k6 and JMeter," Grafana Labs, Jan. 2021
[7] T. Koot, "k6 vs. JMeter," LinkedIn, Oct. 2021
[8] B. Roy, "JMeter vs k6," TestVagrant, Dec. 2022
[9] "JMeter vs. Locust," PFLB, Mar. 2025

Modern Connectivity Enables New Approaches

z/OS systems now offer extensive REST APIs:

- **z/OSMF** - System management and automation
- **z/OS Connect** - RESTful access to CICS and IMS
- **Zowe** - Open-source mainframe framework
- **Modern tooling** - py3270, Ansible, ZOAU, Golang on z/OS [10,11,12,13]

The convergence: Modern tools can leverage these capabilities to drive comprehensive testing.

[10] "py3270: Python interface to x3270," IBM GitHub, 2025
[11] "tnz: Tn3270 to Z Python library," IBM GitHub, 2025
[12] "IBM Z Open Automation Utilities," IBM, 2024
[13] "IBM Open Enterprise SDK for Go," IBM, 2020

Two Industry-Proven Tools, Adapted

🦗 **Locust** (Python) - Used by EA/DICE, AWS, Learnosity

- Extended with py3270 for 3270 terminal automation
- MIT License
- **OSS contribution:** locust-plugins PR #206

🦎 **k6** (Golang/ES6) - Used by GitLab, JPMorgan Chase, Grafana Labs

- Ported to run natively on z/OS UNIX System Services
- GNU Affero General Public License
- **OSS contribution:** k6 PR #2892

Why adapt? Leverage proven tools with massive ecosystems and active communities. [16,17]

[16] "k6 Testimonials," k6.io, 2025
[17] V. Ravi, "Testing shift left observability with the Grafana Stack, OpenTelemetry, and k6," Grafana ObservabilityCON, 2021

Technical Advantages of Modern Tools

Key benefits over legacy approaches:

- 📈 **Scale** - Millions of concurrent users per machine [14,15]
- ⚡ **Efficiency** - 10-30x better resource utilization [6,7]
- 🔀 **Flexibility** - Dynamic patterns, distributed testing, realistic scenarios
- 📊 **Observability** - Real-time metrics, live dashboards, custom exporters
- 🤝 **Open source** - Zero licensing costs, community-driven, auditable code

Battle-tested by major enterprises worldwide.

[14] "Locust documentation," Locust.io, 2025
[15] "k6 documentation," k6.io, 2025
[6] N. van der Hoeven, "Comparing k6 and JMeter," Grafana Labs, Jan. 2021
[7] T. Koot, "k6 vs. JMeter," LinkedIn, Oct. 2021

Two Testing Patterns, Zero Target-Side Agents

External control pattern (Locust + py3270):

```
Workstation/CI-CD → HTTP    → z/OSMF, CICS, Zowe
                  → Telnet  → 3270 terminals
                  → SSH/FTP → z/OS UNIX System Services
```

Native execution pattern (k6 on z/OS):

```
z/OS UNIX System Services → localhost → z/OSMF, CICS, Zowe
```

No persistent agents required. Leverages existing protocols and infrastructure.

Before: STL via IBM TPNS / WSim

```
tso_logon: msgtxt
userid = 'TESTUSER'
password = '****************'

wait until onin substr(ru,1,1) >= '00'x
if substr(message_area,100,9) = 'LOGON ==>' then do
  type '11C540'x||userid
  transmit
  wait until onin substr(ru,1,1) >= '00'x
  type '11C640'x||password||'1DC0'x
  transmit
end

wait until onin substr(ru,1,1) >= '00'x
if substr(message_area,1,5) = 'READY' then do
  type 'SDSF'
```

Proprietary syntax • Hex literals • ITPSTL translator

Locust Example: Session Setup

```
class MainframeUser(tn3270User):
    def on_start(self):
        self.client.connect(user=self.user, password=self.passw)

        # Automate TSO logon sequence
        self.client.string_wait("Application")
        self.client.send_command("TSO")
        self.client.string_wait("ENTER USERID")
        self.client.send_command(self.user)
        self.client.string_wait("Password")
        self.client.send_command(self.passw)
        self.client.string_wait("READY")

        # Enter SDSF
        self.client.send_command("SDSF")
```

Complete 3270 automation: TSO logon → SDSF entry → Config.

Locust Output: Session Initialization

```
[14:11:25] swarm6: Waiting for READY prompt
[14:11:25] swarm6: Successfully logged on!
[14:11:25] swarm6: Current screen text:
  *                                                         *
  *  WELCOME TO THE MAINFRAME TESTING ENVIRONMENT           *
  *  SYSTEM STATUS: OPERATIONAL                             *
  *                                                         *
  *---------------------------------------------------------*
 READY

[14:11:25] swarm6: Entering SDSF
[14:11:25] swarm6: Initialization complete
```

Real 3270 screen capture shows automated TSO logon success.

Locust Example: Test Execution

```python
class MainframeUser(tn3270User):
    wait_time = between(1, 3)  # Realistic think time

    @task(3)  # 3x weight - most common operation
    def display_active(self):
        self.client.send_command("DA")

    @task(2)  # 2x weight
    def output_queue(self):
        self.client.send_command("O")

    @task(2)  # 2x weight
    def hold_queue(self):
        self.client.send_command("H")
```

Weighted tasks mirror production usage patterns.

Locust Output: Concurrent Testing

```
[14:11:26] swarm8: Sending H command (Hold queue)
[14:11:26] swarm9: Sending RES command (Display resources)
[14:11:27] swarm7: Sending O command (Output Queue)
[14:11:27] swarm6: Sending H command (Hold queue)
[14:11:28] swarm10: Sending H command (Hold queue)
[14:11:28] swarm9: Sending JES command (JES subsystem)
[14:11:28] swarm8: Sending DA command (Display Active jobs)
[14:11:29] swarm7: Sending JES command (JES subsystem)
[14:11:30] swarm10: Sending RES command (Display resources)
[14:11:30] swarm6: Sending H command (Hold queue)
[14:11:30] swarm8: Sending DA command (Display Active jobs)
```

5 concurrent virtual users executing weighted SDSF commands.

Before: JMeter Java DSL (2024)

```
import static us.abstracta.jmeter.javadsl.JmeterDsl.*;
import java.time.Duration;

public class ZosmfJobTest {
    @Test
    public void testJobSubmission() throws Exception {
        String jcl = "//TESTJOB JOB (ACCT)\n//STEP1 EXEC PGM=IEFBR14";

        var stats = testPlan(
            threadGroup().rampToAndHold(10, Duration.ofSeconds(30),
                                        Duration.ofMinutes(5)),
            httpDefaults().url("https://zosmf.example.com")
                .header("Authorization", "Basic ${__base64...}"),
            httpSampler("Submit Job").put("/zosmf/restjobs/jobs")
                .body(jcl)
                .children(jsonExtractor("jobid", "jobid")),
            whileController("${__groovy(vars.get('jobStatus')...)}",
                httpSampler("Check").get("/zosmf/restjobs/jobs/${jobname}/${jobid}")
```

JUnit harness • Maven/Gradle • Builder verbosity • Nested complexity

k6 Example: Native z/OS Execution

```
import http from 'k6/http';
import { check } from 'k6';

export const options = { vus: 30, duration: '1m' };

export default function() {
  const jcl = "//TESTJOB JOB (),MSGCLASS=H\n" +
              "// EXEC PGM=IEFBR14";

  const res = http.put(
    'https://localhost:443/zosmf/restjobs/jobs', jcl
  );

  check(res, { 'job submitted': (r) => r.status === 201 });
}
```

Simple JavaScript drives mass job submission via z/OSMF REST API.

k6 Output: High-Volume Testing

```
running (0m57.1s), 30/30 VUs, 999 iterations


[VU8]  Submitted: TESTJOB1/JOB00698
[VU6]  Submitted: TESTJOB2/JOB00687
[VU20] Submitted: TESTJOB3/JOB00686
[VU14] Submitted: TESTJOB5/JOB00669
[VU25] Submitted: TESTJOB2/JOB00662


✓ job submitted
✓ http_req_duration < 500ms
```

999 successful job submissions in 57 seconds from z/OS UNIX System Services.

Production Deployments at IBM Z

Three environments demonstrate real-world value:

- ☁️ **Wazi as a Service** - Locust + z/OSMF for cloud-based testing
- 🐳 **zCX Testing** - Distributed load testing across zCX instances
- ♾️ **Customer Simulation** - k6 running natively on z/OS UNIX System Services for 24/7 synthetic load

Today: Modern open-source tooling at mainframe scale.

AI-Powered Test Case Generation

```
from langchain_ibm import ChatWatsonx

import requests


# Fetch z/OSMF OpenAPI spec (180 endpoints)

response = requests.get("https://example.mainframe.com:443/zosmf/api/docs", auth=("<username>", "<password>"), verify=False)


spec = response.json()


llm = ChatWatsonx(model_id="ibm/granite-3-8b-instruct")


prompt = f"""Given z/OSMF API with {len(spec['paths'])} endpoints,\

generate a Locust load test with 50 users performing file operations \

in /home/locust-user using the restfiles/fs endpoints.""" \


test_code = llm.invoke(prompt)
```

AI + Load Testing: Spec analysis • Intelligent test generation • Edge case discovery

AI-Generated Test Output

```
from locust import HttpUser, task, between
import random, string

class ZosMFFileUser(HttpUser):
    wait_time = between(1, 2)

    @task
    def create_file(self):
        filename = ''.join(random.choices(
            string.ascii_letters + string.digits, k=10))
        self.client.post(
            f"/zosmf/restfiles/fs/{{filename}}.txt",
            json={{"content": "Test data"}},
            headers={{"Authorization": "Bearer <token>"}}
        )

    @task
    def read_file(self):
        filename = ''.join(random.choices(
            string.ascii_letters + string.digits, k=10))
        self.client.get(f"/zosmf/restfiles/fs/{{filename}}.txt")
```

Result: LLM generated functional test scaffolding with proper Locust structure, randomized data, and authentication from 180 API endpoints.

The Future: Intelligent DevOps

Tomorrow: Automated regression + manual verification + randomized testing + AI-assisted test generation.

This multi-modal testing strategy aligns with current DevOps research on quality assurance automation.

Modern tools. Mainframe scale. Radical efficiency.

Getting Started: Open Source & Accessible

```
# Install tools

pip install locust locust-plugins py3270

go install go.k6.io/k6@latest
```

Read more on Medium @msradam:

- "Swarming Stressed Servers"
- "Ticks by Telnet"
- "Go-ing Native"

Slide deck & code: github.com/msradam/swarm-savvy-mcc-2025

References

Load Testing Failures:

[1] "Taylor Swift–Ticketmaster controversy," Wikipedia, 2025 • [2] "Nintendo Switch 2 Pre-Orders," Game Informer, Apr. 2025 • [3] "TSB Bank Failure," iceDQ, Jan. 2025

Mainframe Statistics:

[4] "9 Mainframe Statistics," Precisely, Sep. 2024 • [5] "Banks & Financial Data," Bank Systems & Technology

Tool Comparisons:

[6] N. van der Hoeven, "Comparing k6 and JMeter," Grafana Labs, Jan. 2021 • [7] T. Koot, "k6 vs. JMeter," LinkedIn, Oct. 2021 • [8] B. Roy, "JMeter vs k6," TestVagrant, Dec. 2022 • [9] "JMeter vs. Locust," PFLB, Mar. 2025

z/OS Open Source Tools:

[10] "py3270: Python interface to x3270," IBM GitHub, 2025 • [11] "tnz: Tn3270 to Z," IBM GitHub, 2025 • [12] "IBM Z Open Automation Utilities," IBM, 2024 • [13] "IBM Open Enterprise SDK for Go," IBM, 2020

Documentation:

[14] "Locust documentation," Locust.io, 2025 • [15] "k6 documentation," k6.io, 2025 • [16] "k6 Testimonials," k6.io, 2025 • [17] V. Ravi, "Testing shift left observability with the Grafana Stack, OpenTelemetry, and k6," Grafana ObservabilityCON, 2021

Questions?

Adam Munawar Rahman
Staff Software Developer @ IBM
M.S. Computer Engineering @ NYU Tandon

🌐 adamr.io
📧 Medium: @msradam
💻 github.com/msradam

Modern tools. Mainframe scale. Radical efficiency.