# Approach to Performance Testing Without SLAs or KPIs (On-Premises) Using Access Logs

When conducting **performance testing in an on-premises environment without SLAs, predefined KPIs, or business insights**, **access logs** become the **primary data source** for estimating **realistic workload models, user concurrency, request patterns, and system behavior**.

A structured, **data-driven** approach helps establish **peak load conditions, response times, error trends, and resource consumption**, which can then drive **hardware sizing, workload planning, and scalability testing**.

---

**Step 1: Extract and Analyze Access Logs in Detail**

**Objective: Derive User Traffic Patterns, Usage Trends & System Behavior from Logs**

Since we **do not have predefined SLAs or expected user loads**, analyzing **real system traffic** through **access logs** is **critical**. Access logs can provide:

1. **Request timestamps** – Helps in identifying **peak vs. non-peak traffic hours**.
2. **Requested URLs & Endpoints** – Determines **most frequently accessed** and **heavy-processing transactions**.
3. **HTTP Methods (GET, POST, PUT, DELETE)** – Helps understand **API usage patterns**.
4. **Response Codes (200, 400, 500, etc.)** – Helps **identify failed transactions**.
5. **Response Time Metrics** – Helps in setting **baseline expectations for latency**.
6. **IP Addresses & User-Agent Headers** – Helps determine **unique users, device types (mobile/web), and bot traffic.**

---

**1.1 Extracting Key Metrics from Access Logs**

To extract meaningful data, we can use **Linux commands, Python scripts, or tools like ELK (Elasticsearch, Logstash, Kibana), Splunk, or Grafana**.

**Example Log Format (Apache/Nginx)**

```
192.168.1.10 - - [13/Feb/2024:14:23:04 +0000] "GET /api/account/balance HTTP/1.1" 200 354 "-" "Mozilla/5.0"
```

- **192.168.1.10** → Client IP (can be used to track unique users).
- **[13/Feb/2024:14:23:04]** → Request timestamp (helps identify peak traffic).

- **GET /api/account/balance** → API endpoint (identifies frequently used operations).
- **200** → HTTP status code (helps track success/failure rate).
- **354 ms** → Response time (used for latency calculation).
- **Mozilla/5.0** → User-Agent (identifies whether the request is from mobile/web).

---

### 1.2 Log Analysis Using Linux Commands

#### 1.2.1 Identify Peak Load Hours (Hourly Request Count)

```
awk '{print $4}' access.log | cut -d: -f2 | sort | uniq -c | sort -nr
```

- This command extracts timestamps, counts requests per hour, and sorts them in descending order.
- **Use Case:** Determines when the application experiences the highest load.

#### 1.2.2 Find Most Frequently Accessed APIs/Endpoints

```
awk '{print $7}' access.log | sort | uniq -c | sort -nr | head -20
```

- This command extracts and counts unique endpoints.
- **Use Case:** Helps determine which APIs require **performance optimization**.

#### 1.2.3 Calculate Average Response Time

```
awk '{print $NF}' access.log | sort -n | awk '{count++; sum+=$1} END {print "Avg Response Time:", sum/count}'
```

- Extracts the last column (response time), sorts values, and calculates the **average response time**.
- **Use Case:** Provides **real latency baselines** to define response time KPIs.

#### 1.2.4 Identify HTTP Error Rate (4xx/5xx Failures)

```
awk '$9 ~ /^[45]/ {print $9}' access.log | sort | uniq -c | sort -nr
```

- Extracts status codes **starting with 4xx (client errors) or 5xx (server errors)**.
- **Use Case:** Helps detect API failures and stability issues.

---

### 1.3 Key Observations From Log Analysis

| Metric | Value from Logs | Insights Derived |
|---|---|---|
| **Peak Traffic Hours** | 10 AM - 1 PM | Most users are active in this window. |

| | | |
|---|---|---|
| **Peak Requests Per Second (RPS)** | 400 RPS | Load test should **simulate at least 500 RPS**. |
| **Most Accessed API** | /api/search | This API should be **stress-tested extensively**. |
| **Average Response Time** | 1.5 sec | Define KPI target as **< 2 sec**. |
| **HTTP 5xx Error Rate** | 0.8% | Must **reduce failure rate to < 0.5%**. |

---

**Step 2: Estimate Load and Concurrency from Logs**

**Objective: Use Access Logs to Derive Realistic Load Patterns**

- **Calculate Peak and Off-Peak Concurrency**
  - **Concurrent Users = (Total Requests in Peak Hour) / (Avg Request Time + Think Time)**
  - Example:
    - **Peak hour requests** = 36,000
    - **Avg response time** = 1.5s
    - **Think time** (assumed) = 10s
    - **Requests per user session** = 20
    - **Session duration** = 200s (3.3 min per session)
    - **Total sessions per hour** = 36,000 / 20 = 1,800
    - **Concurrent Users** = (1,800 * 200) / 3,600 = **100-150 users (realistic)**

**2.1 Workload Model Based on Logs**

| Time Slot | Estimated % Traffic | Concurrent Users |
|---|---|---|
| 9 AM - 11 AM | 25% | 1,800 |
| 11 AM - 2 PM | 35% | 2,500 |
| 2 PM - 5 PM | 20% | 1,400 |
| 5 PM - 8 PM | 15% | 1,000 |
| 8 PM - 11 PM | 5% | 400 |

---

**Summary**

✅ **Access logs are the primary data source for deriving user load, API usage, and performance trends.**
✅ **Log analysis provides a realistic estimate of concurrency, RPS, and peak load hours.**
✅ **Key test scenarios are derived based on real system interactions and latency trends.**
✅ **Workload planning uses extracted request rates, peak periods, and concurrency estimates.**
✅ **A structured approach ensures reliable, scalable performance testing, even without SLAs.**

🚀 #PerformanceTesting #PerformanceEngineering 🚀