Scalability
The ability to handle increasing load efficiently **without performance issues**.

Load:
- Users
- Traffic
- Data, etc.

How Can We Detect Increasing Load?
1) **Monitoring Tools:**
   - **Grafana** → Dashboard for visualizing system metrics.
   - **Prometheus** → Query and collect system metrics.
2) **Log Analysis Tools:**
   - **Elasticsearch** → Application logging and search.
   - **Logstash** → Collect, parse, and transform logs.
   - **Kibana** → Data visualization and log analysis.
3) **Cloud Monitoring Solutions:**
   - **AWS CloudWatch** → Collect and analyze metrics for AWS-hosted applications.
   - **GCP Stackdriver** → Monitor and manage GCP-hosted applications.
4) **Tracing Tools:**
   - **Zipkin** → Collects timing data to troubleshoot latency problems.

A System is Considered Scalable if It Can Handle Increasing Load Efficiently Without Performance Issues.

Key Metrics to Detect Load Changes:
1) System Specs:
- **CPU Utilization (%)** → High CPU usage indicates increased workload.
- **Memory Usage (%)** → High memory usage can indicate increased load, memory leaks, or inefficient resource use.

2) Web Traffic:
- **RPS (Requests Per Second)** → A spike in HTTP/gRPC requests indicates rising system demand.
- **Response Time - Latency (ms)** → If API response time increases, the system is struggling to handle the load.

3) Application Performance:
- **Error Rate (%)** → An increase in `HTTP 500 Internal Error / HTTP 503 Service Unavailable` indicates high load or failures.

4) Database Performance:
- **DB Connections Per Second / Queries Per Second** → A high number of queries or connections can cause slow responses or database bottlenecks.

5) Message Broker Load:
- **Queue Length (Kafka / RabbitMQ, etc.)** → Long queues indicate the system cannot process messages fast enough, signaling a need for more consumers or better processing power.

6) JVM Performance (For Java Applications):
- **Thread Pool Saturation** → If worker threads are maxed out, new requests get delayed.