Scaling & Load Balancing in Microservices

Agenda

- Microservices & Scaling Basics
- Vertical vs Horizontal Scaling
- Load Balancing Concepts
- Load Balancing in Microservices
- Tools & Technologies (2025)
- Demo: Node.js with Local Load Balancing
- Summary & Q&A

Introduction to Microservices

- Definition
  - Small, independent services communicating over APIs
- Benefits
  - Scalability, Fault Isolation, Flexibility
- Challenges
  - Distributed nature, network latency, deployment complexity

Why Scaling is Needed

- User base is unpredictable
- Seasonal spikes (e.g., Black Friday, IPL streaming)
- Performance & SLA requirements
- Prevent downtime under heavy load

Amazon Prime Day Traffic Spike

- **Prime Day 2023** recorded **375 million items sold globally** in just **48 hours**.
- The event caused a **~4× increase in concurrent traffic** compared to normal days.
- Certain services saw **extreme, uneven load**:
  - **Product Search API** handled **over 80,000 requests per second** at peak.
  - **Checkout & Payment microservices** spiked by **6×** during flash sale minutes.
  - **Recommendation engine** (ML-powered) faced **3× normal load**.

Types of Scaling

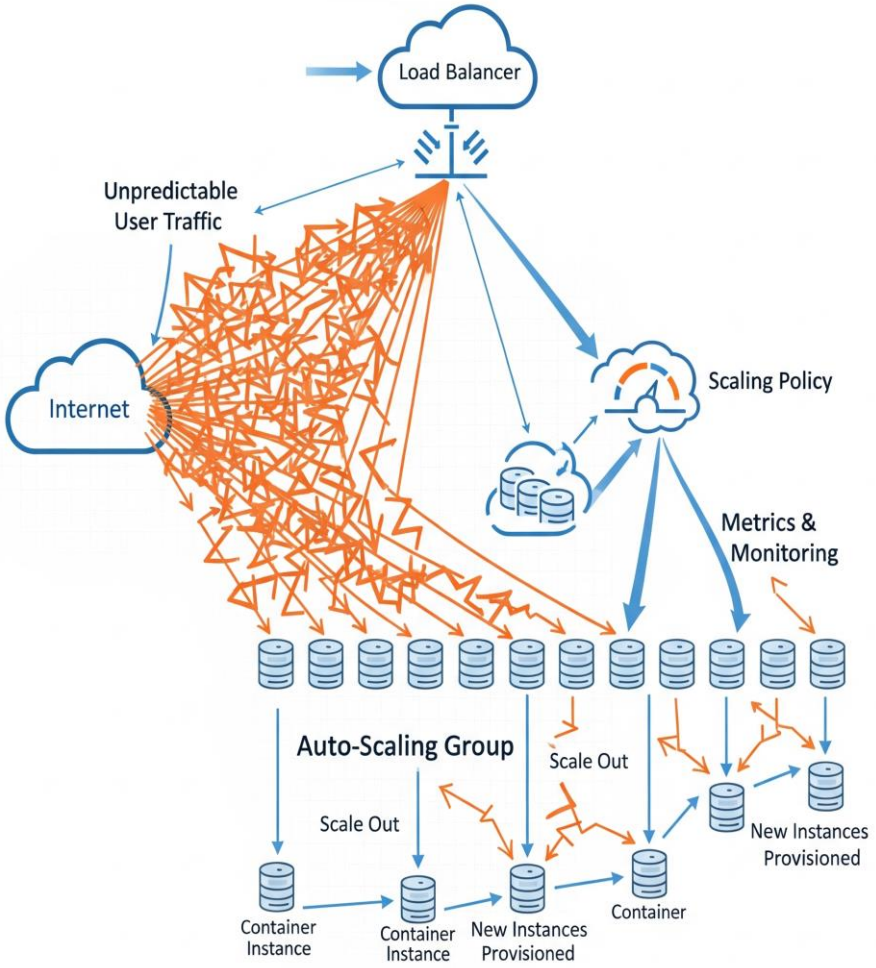
- **Vertical Scaling**
  - Add CPU/RAM to same machine
  - Pros: Simple, no code changes
  - Cons: Hardware limits, single point of failure
- **Horizontal Scaling**
  - Add more instances/machines
  - Pros: Fault-tolerant, cheaper long term
  - Cons: Needs load balancing, distributed stat

Auto Scaling in Microservices

- Cloud-native auto scaling services
  - AWS Auto Scaling Groups
  - Azure Virtual Machine Scale Sets
  - Kubernetes Horizontal Pod Autoscaler (HPA)
- Triggered by metrics
  - CPU usage
  - Request count
  - Custom business metrics

Load Balancing Basics

- Definition: Distributing traffic across multiple servers
- Goals: Improve responsiveness, availability, resource utilization
- Strategies: Round Robin, Least Connections, IP Hash



Strategies

- Round Robin – Equal rotation
- Least Connections – Send to least busy server
- IP Hash – Consistent routing based on client IP
- Weighted Round Robin – More traffic to powerful servers

Load Balancing in Microservices

- Service discovery integration
- Dynamic instance registration/deregistration
- Examples:
  - Nginx,
  - HAProxy,
  - Envoy,
  - Kubernetes Ingress

Service Discovery Tools

- Consul: Key-value store, health checks, DNS interface
- Eureka: Netflix OSS service registry
- Kubernetes DNS: Internal service discovery

Scaling in Kubernetes

- Horizontal Pod Auto-scaler (HPA)
- Cluster Auto-scaler
- Resource Requests & Limits

Cloud-native Scaling Tools

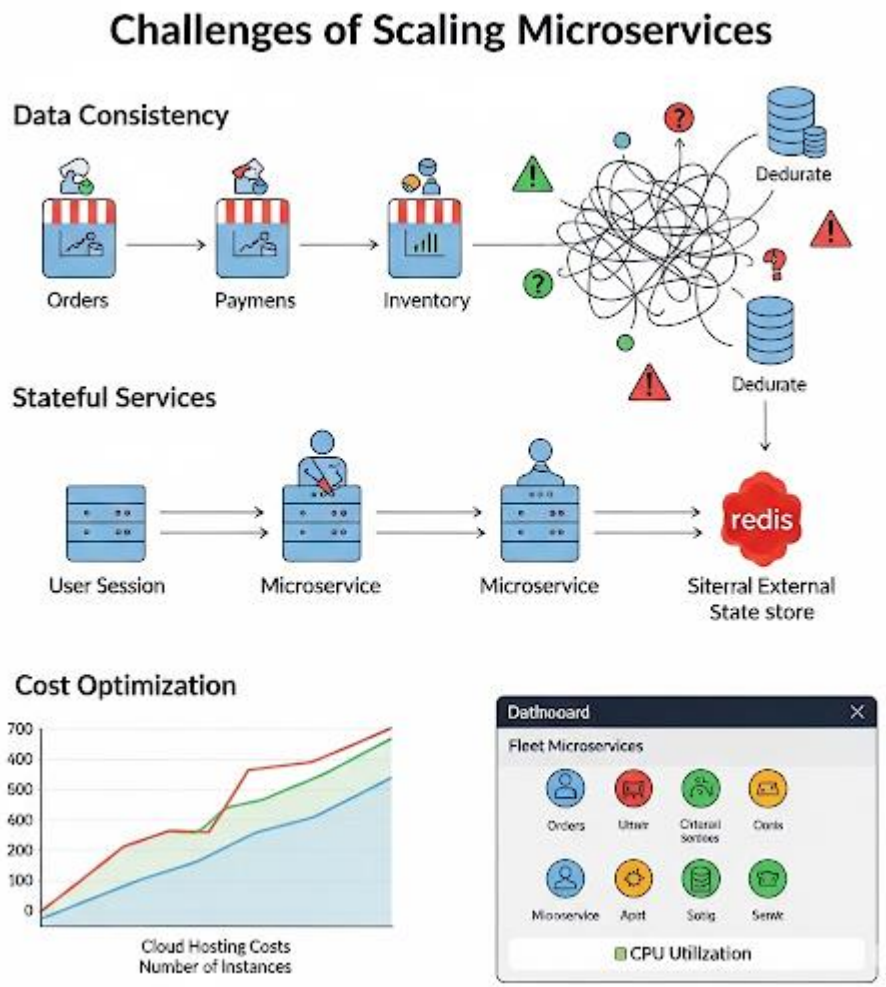
- AWS Auto Scaling Groups
- GCP Instance Groups
- Azure VM Scale Sets

Load Balancing in Cloud Platforms

- AWS Elastic Load Balancer (ALB/NLB)
- GCP Load Balancer
- Azure Load Balancer

Challenges in Scaling

- Data consistency
- Stateful services
- Cost optimization



Best Practices

- Stateless service design
- Monitoring & alerting
- Graceful degradation

Demo Plan

- Node.js microservice with frontend
- Run multiple instances
- Configure Nginx load balancer
- Test with load testing tool (e.g., Apache JMeter)

Demo Architecture

- Frontend: React
- Backend: Node.js + Express
- Load Balancer: Nginx
- Service Discovery: Consul

Demo Step-by-Step

- Build Node.js service
- Create Dockerfile
- Run multiple containers
- Configure Nginx for round-robin
- Verify scaling

Latest Tech Stack

- Node.js 20+
- React 18+
- Docker 25+
- Consul 1.16+
- Nginx 1.25+

Monitoring Tools

- Prometheus + Grafana
- ELK Stack
- Datadog

Feature	**Scaling**	**Load Balancing**
Definition	Increasing or decreasing the capacity of your system to handle load.	Distributing incoming requests evenly across multiple instances or servers.
Goal	To handle **more requests** by adding computing power (scale up) or more instances (scale out).	To ensure **no single instance** is overwhelmed and traffic is handled efficiently.
Types	- **Vertical Scaling (Scale-Up):** Adding more CPU/RAM to an instance. - **Horizontal Scaling (Scale-Out):** Adding more instances of a service.	- **DNS Load Balancing** (Round-robin, weighted). - **Application Load Balancer** (Layer 7). - **Network Load Balancer** (Layer 4).
When Needed	When your current system **cannot handle peak traffic** or **performance degrades**.	When you **already have multiple instances** running and need to split traffic.

Feature	**Scaling**	**Load Balancing**
Example in Microservices	Adding 5 more pods for the payment-service in Kubernetes when CPU > 80%.	Routing checkout requests evenly to all payment-service pods so no single pod is overloaded.
Cloud/DevOps Tools	- Kubernetes Horizontal Pod Autoscaler (HPA) - AWS Auto Scaling Groups - Azure VM Scale Sets - Google Cloud Instance Groups	- AWS Elastic Load Balancing (ELB) - Nginx / HAProxy - Kubernetes Ingress Controller
Key Metrics	CPU/Memory utilization, request rate (RPS), queue length, response time.	Response time per instance, request distribution, health checks.
Relation	Scaling **adds capacity**; load balancing **manages traffic across capacity**.	Load balancing **needs capacity**; scaling **provides that capacity**.

Conclusion

- Scaling & load balancing are key for performance
- Choose right tools for your architecture
- Automate wherever possible

Q&A

- Open discussion