

# **NGINX WORKSHOP: From Scaling Problems to High Availability**

**DevOps Bootcamp**

BECHAR Walid

# Workshop Overview

**"From Scaling Problems → Reverse Proxies → Advanced NGINX → HA Architecture"**

In this workshop, we'll explore:

- The challenges of scaling applications with NGINX
- How reverse proxies solve scaling problems
- Using NGINX as a powerful reverse proxy and load balancer
- Advanced NGINX features and configurations
- Building robust architecture patterns with and without orchestration
- Route-based proxying and service composition

# 1. The Scaling Problem

## Why Scaling Exists

- Traffic spikes and concurrency challenges
- CPU/RAM exhaustion under load
- Server crashes and downtime
- Poor user experience with timeouts and errors

## Real-World Analogy

Think of a single cashier at a busy store:

- Customers form long queues
- Some give up and leave
- One point of failure

## What Users Experience

- Request timeouts
- 500 Internal Server Errors
- Slow page loads
- Complete service unavailability

## 2. Types of Scaling

### Vertical Scaling (Scale Up)

- Add more CPU/RAM to the same machine
- **Pros:** Fast implementation, simple
- **Cons:** Limited resources, expensive, SPOF (Single Point of Failure)

### Horizontal Scaling (Scale Out)

- Add more instances/containers
- Foundation for modern DevOps and microservices
- **Pros:** Almost unlimited scaling, redundancy

### 3. Approaches to Scaling

#### Using Managed Cloud Infrastructure

- Cloud load balancers and auto-scaling services
- Platform handles scaling automatically
- Health checks and instance management
- **Pros:** Easy to implement, managed service
- **Cons:** Expensive, less control over infrastructure

#### Scaling Manually on Your VPS

- You control everything (infrastructure + scaling logic)
- Budget-friendly approach for learning
- Requires manual design of:
  - Load balancing mechanisms
  - High availability (HA) patterns
  - Failover procedures
  - Monitoring solutions

## 4. The Need for Load Distribution

### The Missing Piece

- Adding multiple instances isn't enough
- You need something to distribute traffic among them
- This is where reverse proxies come in

### What Problems Need Solving

- How do requests reach the right instance?
- What happens when an instance fails?
- How do users access multiple instances through one URL?
- How do we balance the load evenly?

### Enter Reverse Proxies

- Single entry point for your application
- Distributes requests to backend instances
- Handles health checks and failover

## 5. What is NGINX?

### Core Capabilities

- **L7 Reverse Proxy:** Application layer routing
- **Load Balancer:** Traffic distribution
- **Cache:** Content caching for performance
- **Static Server:** Serve static files efficiently
- **API Gateway Lite:** Route API requests
- **SSL Terminator:** Handle HTTPS connections

### Why NGINX is Popular

- High performance and low resource usage
- Extensive documentation and community
- Battle-tested in production environments
- Flexible configuration options

## 6. NGINX: Load Distribution, Not Scaling

### Important Distinction

- **NGINX does NOT provide scaling by itself**
- NGINX distributes high load across existing instances
- For NGINX to be effective, you need multiple application instances

### What NGINX Actually Does

- Distributes incoming requests across multiple backend servers
- Balances load to prevent any single server from being overwhelmed
- Provides health checks to route traffic only to healthy instances

### The Real Scaling Challenge

- NGINX can efficiently distribute traffic to 3 instances
- But what if you need 10, 50, or 100+ instances?
- Manual creation and management of instances doesn't scale

### This Leads To...

The need for solutions that can automatically add/remove nodes based on demand!

## 7. How NGINX Solves Distribution Problems

### Request Distribution

- NGINX distributes requests to multiple backend instances
- Various load balancing algorithms available

### Health Checks

- Can detect failed backends and avoid routing to them
- Automatic failover to healthy instances

### Single Access Point

- Users access a single endpoint
- NGINX handles routing to appropriate backend

### Session Management

- Sticky sessions to maintain user state
- Consistent user experience

### But Remember

- NGINX handles traffic distribution, not instance creation
- You still need to provision additional instances manually

## 8. Load Balancing Techniques with NGINX

### Round Robin

- Cycles through servers sequentially
- Simple and fair distribution

### Least Connections

- Sends requests to server with fewest active connections
- Better for long-running connections

### IP Hash

- Uses client IP to determine target server
- Ensures session persistence

### Weighted Strategies

- Assigns different weights to servers
- Accommodates different server capabilities

### NGINX Native Support

- All major algorithms supported natively
- Custom configurations possible

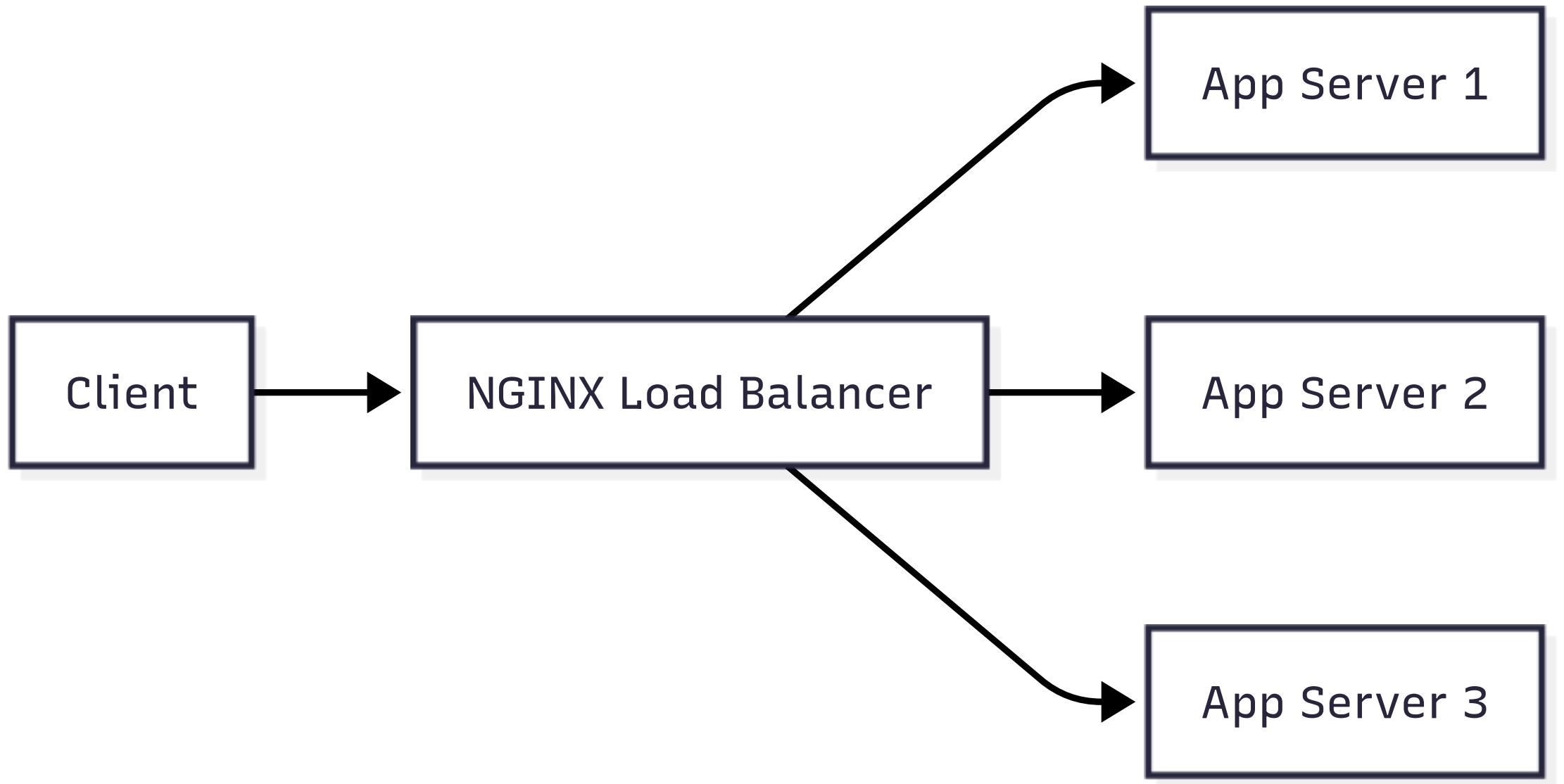
## 9. Intro to Reverse Proxies & Load Balancers

### Reverse Proxy

- Sits between clients and servers
- Single entry point for applications
- Hides internal architecture complexity
- Can provide security, caching, SSL termination

### Load Balancer

- Distributes traffic between multiple instances
- Can be **L4 (Transport Layer)** or **L7 (Application Layer)**
- Ensures high availability
- Provides health checks and failover



# 10. How to Use NGINX

## Basic Reverse Proxy Config

```
upstream backend {  
    server app1:8001;  
    server app2:8002;  
    server app3:8003;  
}  
  
server {  
    listen 80;  
    location / {  
        proxy_pass http://backend;  
    }  
}
```

## Key Features

- Load balancing across containers/instances
- Health checks for backend servers
- SSL termination
- Rate limiting
- Request/response logging
- Custom upstream behavior

## 11. NGINX with and without Orchestration

### Standalone NGINX Benefits

- Simple setup and management
- No additional complexity from orchestration tools
- Perfect for small to medium applications
- Lower resource overhead

### When to Add Orchestration

- Managing many application instances
- Need for auto-scaling based on metrics
- Self-healing capabilities required
- Complex deployment patterns

### Key Point

NGINX works perfectly well without Swarm, Kubernetes, or other orchestration tools!

## 12. Horizontal Scaling with Multiple Instances

### Adding Application Instances Manually

Running multiple app instances:

- Port 8001
- Port 8002
- Port 8003
- ...

### NGINX Configuration

```
upstream backend {
    server app1:8001;
    server app2:8002;
    server app3:8003;
}

server {
    listen 80;
    location / {
        proxy_pass http://backend;
    }
}
```

## **The Limitation**

- You must manually add/remove servers from the configuration
- No automatic scaling based on load
- Requires manual intervention to handle traffic spikes

# 13. Route-Based Reverse Proxying

## Advanced NGINX Feature

NGINX can route requests to different backends based on URL paths:

### Example Configuration

```
server {
    listen 80;

    # Route static assets to CDN
    location /static/ {
        proxy_pass http://cdn.example.com;
    }

    # Route API requests to backend services
    location /api/ {
        proxy_pass http://api-backend;
    }

    # Route admin requests to admin service
    location /admin/ {
        proxy_pass http://admin-service;
    }

    # Default route to main application
    location / {
        proxy_pass http://main-app;
    }
}
```

## **14. Service Composition with NGINX**

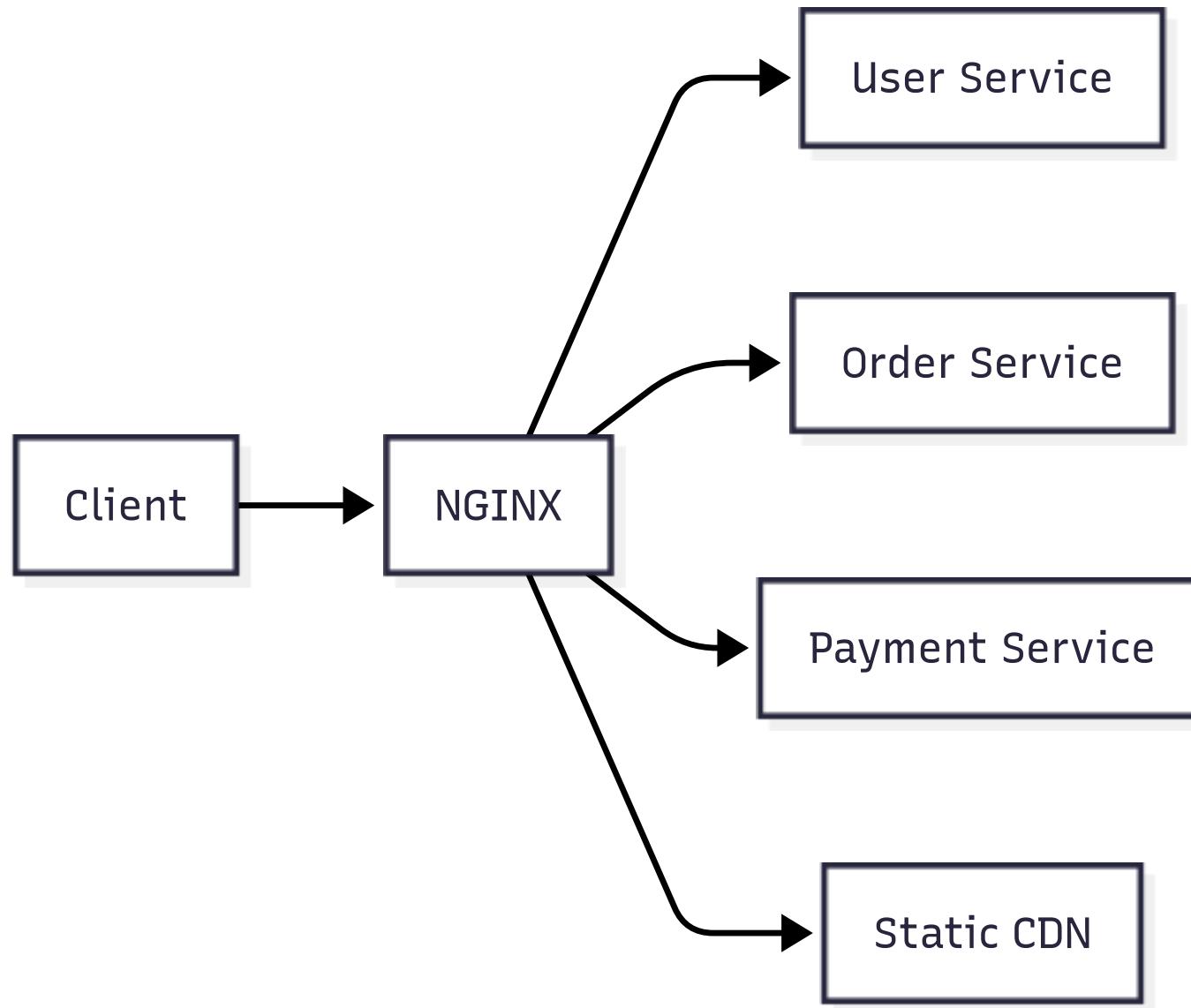
### **API Gateway Pattern**

- Route different paths to different microservices
- Aggregate multiple services behind single domain
- Handle cross-cutting concerns (authentication, rate limiting)

### **Static Content Optimization**

- Route static content (/static, /images, /css) to CDN
- Serve static files directly from NGINX for performance
- Reduce backend server load

## Microservices Architecture



## 15. Solutions for Dynamic Scaling (Adding Nodes)

### The Next Step: Automatic Instance Management

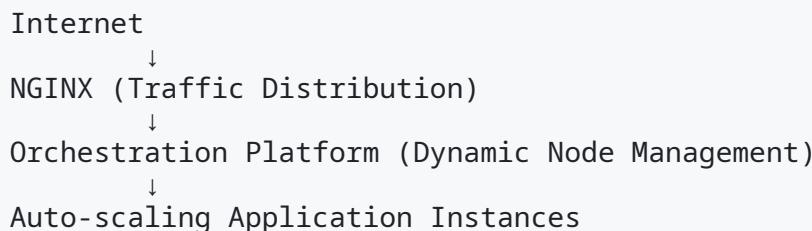
While NGINX handles load distribution, we need solutions to:

- Automatically add more instances when traffic increases
- Remove instances when traffic decreases
- Handle instance failures automatically
- Manage the lifecycle of application nodes

### Orchestration Solutions

- **Docker Swarm**: Native Docker clustering
- **Kubernetes**: Container orchestration platform
- **Cloud Services**: AWS Auto Scaling Groups, GCP Managed Instance Groups

### The Complete Picture



## 16. Load Testing with NGINX

### Testing Scenarios

- Single container through NGINX
- Multiple containers with NGINX load balancing
- Route-based proxying performance

### Tools for Testing

- `hey` (HTTP load generator)
- `ab` (Apache Bench)
- `k6` (Modern load testing)

### Metrics to Compare

- Throughput (requests per second)
- Latency (response times)
- Error rates under load
- Resource utilization

## 17. Advanced NGINX Features

### Performance Tuning

- **Buffers:** Control memory usage for request/response processing
- **Timeouts:** Configure connection and read/write timeouts
- **Caching:** Store frequently accessed content

### Security & Control

- **Rate Limiting:** DDoS protection and traffic control
- **Sticky Sessions:** Ensures session affinity
- **Logging Pipeline:** Integration with ELK stack

### Route-Based Features

- **Path Rewriting:** Modify URLs in transit
- **Header Manipulation:** Add, remove, or modify HTTP headers
- **Authentication:** Handle authentication at the proxy layer

## 18. Architecture Considerations

### When to Use NGINX Standalone

- Small to medium traffic applications
- Simple deployment requirements
- Need for route-based routing
- Cost-sensitive environments

### When to Add Dynamic Scaling Later

- High-traffic applications requiring auto-scaling
- Complex deployment requirements
- Need for service mesh features
- Advanced observability needs

### Key Takeaway

NGINX handles traffic distribution effectively, but when you need to dynamically add/remove nodes based on demand, orchestration tools become valuable additions.

# Advanced Routing Examples

## Content Delivery Optimization

```
server {
    listen 80;

    # Static assets to CDN with caching headers
    location ~* \.(jpg|jpeg|png|gif|ico|css|js)$ {
        proxy_pass http://cdn-backends;
        expires 1y;
        add_header Cache-Control "public, immutable";
    }

    # API requests with rate limiting
    location /api/ {
        limit_req zone=api burst=10 nodelay;
        proxy_pass http://api-servers;
    }

    # WebSocket connections
    location /ws/ {
        proxy_pass http://websocket-servers;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }
}
```

# Workshop Conclusion

## Key Takeaways

- Scaling is essential for modern applications
- NGINX is a powerful load distribution solution
- Route-based proxying enables service composition
- NGINX works perfectly without orchestration tools
- For dynamic scaling, combine NGINX with orchestration platforms

## What You've Learned

- NGINX handles traffic distribution, not instance scaling
- Load balancing across multiple instances
- Route-based proxying for different services
- High availability patterns with NGINX
- When to consider orchestration tools for dynamic node management

**Thank you for attending!**