Nginx and Base Path Routing Fundamentals

A Comprehensive Tutorial Based on Week 03 Multi-Chatbot Architecture

---

1. Introduction to Nginx and Routing

What is Nginx?

Nginx (pronounced "engine-x") is a high-performance web server, reverse proxy, and load balancer. Originally created to solve the C10k problem (handling 10,000+ concurrent connections), Nginx has become one of the most popular web servers worldwide.

Key Capabilities

- **Web Server**: Serve static files efficiently
- **Reverse Proxy**: Forward requests to backend services
- **Load Balancer**: Distribute traffic across multiple servers
- **SSL Termination**: Handle HTTPS encryption/decryption
- **Caching**: Store frequently requested content
- **API Gateway**: Route and manage API requests

Why Nginx for Multi-Application Hosting?

In our Week 03 setup, we use Nginx to host 5 different chatbot applications under a single domain:

```
https://project-1-xx.eduhk.hk/chatbot01/  → Node.js App (Port 3000)
https://project-1-xx.eduhk.hk/chatbot02/  → Node.js App (Port 3001)
https://project-1-xx.eduhk.hk/chatbot03/  → Node.js App (Port 3002)
https://project-1-xx.eduhk.hk/chatbot04/  → Python FastAPI (Port 8001)
https://project-1-xx.eduhk.hk/chatbot05/  → React + Python (Port 8002)
```

This architecture provides:

- **Unified Domain**: Single SSL certificate for all apps
- **Path-Based Routing**: Different applications under different paths
- **Service Isolation**: Each app runs independently
- **Easy Scaling**: Add new applications by adding location blocks

---

2. Nginx Architecture Fundamentals

Master-Worker Architecture

```
Master Process
├── Worker Process 1 (handles requests)
├── Worker Process 2 (handles requests)
├── Worker Process N (handles requests)
└── Cache Manager/Loader Process
```

Configuration Hierarchy

```
Main Context
├── Events Context
├── HTTP Context
│   ├── Server Context 1
│   │   ├── Location Context 1
│   │   ├── Location Context 2
│   │   └── Location Context N
│   └── Server Context 2
└── Stream Context (TCP/UDP)
```

Week 03 Configuration Structure

Our configuration follows this hierarchy:

```
# Main Context - Global directives
events { ... }

http {
    # HTTP Context - HTTP-specific directives
```

```
    server {
        # Server Context - Virtual host for project-1-xx.eduhk.hk
        listen 443 ssl;
        server_name project-1-xx;

        location /chatbot01/ { ... }  # Location Context for Chatbot 01
        location /chatbot02/ { ... }  # Location Context for Chatbot 02
        location /chatbot03/ { ... }  # Location Context for Chatbot 03
        location /chatbot04/ { ... }  # Location Context for Chatbot 04
        location /chatbot05/ { ... }  # Location Context for Chatbot 05
    }
}
```

---

3. Server Blocks and Virtual Hosts

Understanding Server Blocks

A server block defines how to handle requests for a specific server name (domain) and port combination.

Basic Server Block Structure

```
server {
    listen 80;                    # Port to listen on
    server_name example.com;      # Domain name
    root /var/www/html;           # Document root
    index index.html;             # Default files

    location / {
        # Handle requests to this server
    }
}
```

Week 03 Server Blocks Analysis

Our configuration has two server blocks:

1. HTTP Redirect Server (Port 80)

```
server {
    listen 80;
    server_name project-1-xx;
    return 301 https://$host$request_uri;  # Redirect all HTTP to HTTPS
}
```

Purpose: Force HTTPS by redirecting all HTTP requests **Variables Used**:

- `$host`: The server name from the request
- `$request_uri`: The full original request URI including arguments

2. HTTPS Main Server (Port 443)

```
server {
    listen 443 ssl;
    server_name project-1-xx;

    # SSL Configuration
    ssl_certificate /etc/nginx/ssl/dept-wildcard.eduhk/fullchain.crt;
    ssl_certificate_key /etc/nginx/ssl/dept-wildcard.eduhk/dept-
wildcard.eduhk.hk.key;
    ssl_protocols TLSv1.2 TLSv1.3;

    # Location blocks for different applications
    location /chatbot01/ { ... }
    location /chatbot02/ { ... }
    # ... more locations
}
```

Purpose: Handle all HTTPS requests and route them to appropriate applications

Server Name Matching

Nginx matches requests to server blocks using this priority:

1. **Exact Match**: `server_name example.com;`
2. **Wildcard at Start**: `server_name *.example.com;`
3. **Wildcard at End**: `server_name example.*;`
4. **Regular Expression**: `server_name ~^(?<subdomain>.+)\.example\.com$;`
5. **Default Server**: `server { listen 80 default_server; }`

---

4. Location Directive Deep Dive

Location Matching Syntax

```
location [modifier] pattern {
    # Configuration for matching requests
}
```

Location Modifiers (Priority Order)

1. `=` **(Exact Match)** - Highest priority
2. `^~` **(Prefix Match, No Regex)** - High priority
3. `~` **(Case-Sensitive Regex)** - Medium priority

4. `~*` **(Case-Insensitive Regex)** - Medium priority
5. **No Modifier (Prefix Match)** - Lowest priority

Week 03 Location Examples

1. Exact Match for Trailing Slash Redirect

```
location = /chatbot01 {
    return 301 /chatbot01/;
}
```

- **Pattern**: Exactly `/chatbot01`
- **Purpose**: Redirect `/chatbot01` to `/chatbot01/` (with trailing slash)
- **Priority**: Highest (exact match)

2. Prefix Match for Application Routing

```
location /chatbot01/ {
    proxy_pass http://127.0.0.1:3000/;
}
```

- **Pattern**: Starts with `/chatbot01/`
- **Matches**: `/chatbot01/`, `/chatbot01/api`, `/chatbot01/static/style.css`
- **Priority**: Lowest (prefix match)

3. Specific API Endpoint

```
location /chatbot04/chat/completions {
    proxy_pass http://127.0.0.1:8001/chat/completions;
}
```

- **Pattern**: Exact path `/chatbot04/chat/completions`
- **Purpose**: Route specific API endpoint to backend
- **Priority**: Highest (longer prefix wins)

Location Matching Examples

Given these location blocks:

```
location = /chatbot01 { return 301 /chatbot01/; }           # A
location /chatbot01/ { proxy_pass http://127.0.0.1:3000/; } # B
location /chatbot04/chat/completions { ... }                # C
location /chatbot04/ { ... }                                # D
```

Request matching:

- `/chatbot01` → A (exact match)
- `/chatbot01/` → B (prefix match)
- `/chatbot01/login` → B (prefix match)
- `/chatbot04/` → D (prefix match)
- `/chatbot04/chat/completions` → C (longer prefix wins)

---

5. Base Path Routing Concepts

What is Base Path Routing?

Base path routing allows multiple applications to share a single domain by using different URL paths as entry points. Each application gets its own "base path" or "context path".

Traditional vs Base Path Architecture

Traditional Approach (Multiple Domains)

```
chatbot01.eduhk.hk  → App 1
chatbot02.eduhk.hk  → App 2
chatbot03.eduhk.hk  → App 3
```

Disadvantages: Multiple SSL certificates, DNS management, subdomain complexity

Base Path Approach (Single Domain)

```
eduhk.hk/chatbot01/  → App 1
eduhk.hk/chatbot02/  → App 2
eduhk.hk/chatbot03/  → App 3
```

Advantages: Single SSL certificate, unified domain, easier management

Week 03 Base Path Strategy

Our configuration implements base path routing with these patterns:

```
# Pattern 1: Full Proxy (Backend handles everything)
location /chatbot01/ {
    proxy_pass http://127.0.0.1:3000/;
    # Request: /chatbot01/api/health
    # Proxied to: http://127.0.0.1:3000/api/health
}

# Pattern 2: Static Files + API Proxy
```

```
    location /chatbot04/ {
        alias /home/proj07/project-1-xx/chatbot_04_MVPPython/;
        # Serves static files directly from filesystem
    }

    location /chatbot04/chat/completions {
        proxy_pass http://127.0.0.1:8001/chat/completions;
        # API requests go to backend
    }

    # Pattern 3: SPA + API Separation
    location /chatbot05/ {
        alias /var/www/html/chatbot05/;
        try_files $uri $uri/ /chatbot05/index.html;
        # React app with client-side routing
    }

    location /api/v2/ {
        proxy_pass http://127.0.0.1:8002/;
        # Dedicated API prefix
    }
```

URL Rewriting in Base Path Routing

Path Preservation vs Path Modification

Path Preservation (Our Chatbot 01-03):

```
    location /chatbot01/ {
        proxy_pass http://127.0.0.1:3000/;
    }
    # /chatbot01/api/health → http://127.0.0.1:3000/api/health
    # (chatbot01/ is stripped)
```

Path Modification (Alternative):

```
    location /chatbot01/ {
        proxy_pass http://127.0.0.1:3000/chatbot01/;
    }
    # /chatbot01/api/health → http://127.0.0.1:3000/chatbot01/api/health
    # (chatbot01/ is preserved)
```

Base Path Challenges and Solutions

When implementing base path routing in production environments, several common challenges arise. Let's explore each challenge in detail with practical examples from our Week 03 setup and comprehensive solutions.

Challenge 1: Asset Path Resolution 🎨

The Problem in Detail:

Modern JavaScript frameworks like React, Vue, and Angular build their applications assuming they will be served from the root path (`/`). When you run `npm run build`, the build process generates HTML files that reference assets using absolute paths:

```
<!-- React build output assumes root serving -->
<link href="/static/css/main.1234.css" rel="stylesheet">
<script src="/static/js/main.5678.js"></script>
<img src="/static/media/logo.abcd.png" alt="Logo">
```

When served from a base path like `/chatbot05/`, these asset requests become:

- `GET /static/css/main.1234.css` (404 - not found)
- `GET /static/js/main.5678.js` (404 - not found)
- `GET /static/media/logo.abcd.png` (404 - not found)

But the actual files are located at:

- `/var/www/html/chatbot05/static/css/main.1234.css`
- `/var/www/html/chatbot05/static/js/main.5678.js`
- `/var/www/html/chatbot05/static/media/logo.abcd.png`

Real-World Example from Week 03:

In our Chatbot 05 setup, the React app structure looks like this:

```
/var/www/html/chatbot05/
├── index.html
├── static/
│   ├── css/
│   │   └── main.1234.css
│   ├── js/
│   │   └── main.5678.js
│   └── media/
│       └── logo.abcd.png
└── asset-manifest.json
```

Solution 1: Nginx Rewrite Rules (Our Implementation)

```
# Redirect root asset requests to chatbot05 assets
rewrite ^/static/(.*)$ /chatbot05/static/$1 last;
rewrite ^/assets/(.*)$ /chatbot05/assets/$1 last;

location /chatbot05/ {
    alias /var/www/html/chatbot05/;
```

```
    try_files $uri $uri/ /chatbot05/index.html;
    index index.html;
}
```

How it works:

1. Browser requests: `GET /static/css/main.1234.css`
2. Nginx rewrite rule transforms it to: `GET /chatbot05/static/css/main.1234.css`
3. The location block serves the file from: `/var/www/html/chatbot05/static/css/main.1234.css`

Solution 2: Build-Time Configuration

```
# Configure React to build with base path
PUBLIC_URL=/chatbot05 npm run build

# Or using environment variable
REACT_APP_BASE_URL=/chatbot05 npm run build
```

This generates HTML with correct asset paths:

```
<link href="/chatbot05/static/css/main.1234.css" rel="stylesheet">
<script src="/chatbot05/static/js/main.5678.js"></script>
```

Solution 3: Multiple Location Blocks

```
location /chatbot05/ {
    alias /var/www/html/chatbot05/;
    try_files $uri $uri/ /chatbot05/index.html;
}

# Explicit asset handling
location /static/ {
    alias /var/www/html/chatbot05/static/;
    expires 1y;
    add_header Cache-Control "public, immutable";
}

location /assets/ {
    alias /var/www/html/chatbot05/assets/;
    expires 1y;
    add_header Cache-Control "public, immutable";
}
```

Debug Asset Issues:

```
# Check if assets exist
ls -la /var/www/html/chatbot05/static/

# Test asset access directly
curl -I https://project-1-xx.eduhk.hk/static/css/main.css

# Monitor nginx access logs for 404s
sudo tail -f /var/log/nginx/access.log | grep "404"
```

Challenge 2: Trailing Slash Consistency 🔁

The Problem in Detail:

URLs with and without trailing slashes are treated as different resources by web servers. This creates several issues:

1. **SEO Problems**: Search engines see duplicate content
2. **Broken Links**: Internal links may break
3. **Inconsistent Behavior**: Different responses for similar URLs
4. **Cache Issues**: CDNs may cache both versions separately

Real-World Scenarios:

```
# Different behaviors without proper handling:
GET /chatbot01   → 404 Not Found (no location match)
GET /chatbot01/  → 200 OK (matches location /chatbot01/)

# User bookmarks and shares inconsistent URLs:
https://project-1-xx.eduhk.hk/chatbot01    # Bookmark 1
https://project-1-xx.eduhk.hk/chatbot01/   # Bookmark 2
```

Our Week 03 Solution:

```
# Exact match for URL without trailing slash
location = /chatbot01 {
    return 301 /chatbot01/;
}

# Handle all requests with trailing slash
location /chatbot01/ {
    proxy_pass http://127.0.0.1:3000/;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_redirect off;
}
```

Why This Works:

1. **301 Permanent Redirect**: Tells browsers and search engines the canonical URL
2. **Exact Match Priority**: `location =` has highest priority, ensures precise matching
3. **SEO Friendly**: Consolidates page authority to one URL version

Alternative Approaches:

Option 1: Remove Trailing Slashes

```
# Redirect WITH trailing slash to WITHOUT
location = /chatbot01/ {
    return 301 /chatbot01;
}

location /chatbot01 {
    proxy_pass http://127.0.0.1:3000/;
    # ... proxy headers
}
```

Option 2: Dynamic Handling

```
location ~ ^(/chatbot01)/?$ {
    set $upstream http://127.0.0.1:3000;
    proxy_pass $upstream;
    # ... proxy headers
}
```

Testing Trailing Slash Handling:

```
# Test redirect behavior
curl -I http://project-1-xx.eduhk.hk/chatbot01
# Should return: HTTP/1.1 301 Moved Permanently
# Location: /chatbot01/

# Test final destination
curl -I http://project-1-xx.eduhk.hk/chatbot01/
# Should return: HTTP/1.1 200 OK
```

Challenge 3: API Route Conflicts 🔀

The Problem in Detail:

When hosting multiple applications under base paths, API route conflicts become a major issue:

```
# Problematic setup - conflicts possible
location /chatbot01/ {
    proxy_pass http://127.0.0.1:3000/;
    # App 1 might have: GET /api/users
}

location /chatbot02/ {
    proxy_pass http://127.0.0.1:3001/;
    # App 2 might also have: GET /api/users
}

# Global API route - which app should handle this?
location /api/ {
    proxy_pass http://127.0.0.1:????/;  # Conflict!
}
```

Week 03 Solutions Implemented:

Solution 1: App-Specific API Paths

```
# Chatbot 04 - API under app path
location /chatbot04/chat/completions {
    proxy_pass http://127.0.0.1:8001/chat/completions;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_redirect off;

    # Streaming configuration
    proxy_buffering off;
    proxy_cache off;
    proxy_set_header Connection '';
    proxy_http_version 1.1;
    chunked_transfer_encoding off;
}
```

Solution 2: Versioned Global APIs

```
# Chatbot 05 - Dedicated API version
location /api/v2/ {
    proxy_pass http://127.0.0.1:8002/;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_redirect off;

    # Enable streaming
```

```
    proxy_buffering off;
    proxy_cache off;
    proxy_set_header Connection '';
    proxy_http_version 1.1;
    chunked_transfer_encoding off;
}
```

Advanced Conflict Resolution Strategies:

Strategy 1: Namespace by Application

```
location /chatbot01/api/ {
    proxy_pass http://127.0.0.1:3000/api/;
}

location /chatbot02/api/ {
    proxy_pass http://127.0.0.1:3001/api/;
}

location /chatbot03/api/ {
    proxy_pass http://127.0.0.1:3002/api/;
}
```

Strategy 2: Service-Based Routing

```
# User management service
location /api/users/ {
    proxy_pass http://user-service:8001/;
}

# Chat service
location /api/chat/ {
    proxy_pass http://chat-service:8002/;
}

# Analytics service
location /api/analytics/ {
    proxy_pass http://analytics-service:8003/;
}
```

Strategy 3: Header-Based Routing

```
location /api/ {
    set $upstream "";

    if ($http_x_app_version = "v1") {
        set $upstream "http://127.0.0.1:8001";
```

```
    }
    if ($http_x_app_version = "v2") {
        set $upstream "http://127.0.0.1:8002";
    }

    proxy_pass $upstream;
}
```

Challenge 4: Client-Side Routing vs Server-Side Routing 🧭

The Problem in Detail:

Single Page Applications (SPAs) like React use client-side routing, where the browser's URL changes without server requests. This conflicts with nginx's server-side routing:

```
User navigates in browser:
/chatbot05/ → /chatbot05/dashboard → /chatbot05/settings

React handles internally, but if user refreshes:
GET /chatbot05/dashboard → nginx tries to find file → 404
```

Our Week 03 Solution:

```
location /chatbot05/ {
    alias /var/www/html/chatbot05/;
    try_files $uri $uri/ /chatbot05/index.html;
    index index.html;
}
```

How try_files Works:

1. **$uri**: Try to serve exact path as file
2. **$uri/**: Try to serve path as directory with index
3. **Fallback**: Serve `/chatbot05/index.html` (React app takes over)

Advanced SPA Routing:

```
location /chatbot05/ {
    alias /var/www/html/chatbot05/;

    # First, try exact file
    try_files $uri $uri/ @spa_fallback;
    index index.html;
}

# Named location for SPA fallback
location @spa_fallback {
```

```
    rewrite ^.*$ /chatbot05/index.html last;
}

# Handle API routes specifically (don't fallback to SPA)
location /chatbot05/api/ {
    return 404; # Or proxy to backend
}
```

Challenge 5: Base Path in Application Configuration ⚙️

The Problem in Detail:

Applications need to be aware of their base path for:

- Generating correct URLs in responses
- Setting up client-side routing
- Configuring API endpoints
- Managing redirects and cookies

Backend Configuration Examples:

Node.js Express (Chatbot 01-03):

```
const express = require('express');
const app = express();

// Handle base path in application
app.use('/chatbot01', express.static('public'));

// Or use environment variable
const BASE_PATH = process.env.BASE_PATH || '';
app.use(BASE_PATH, routes);

// Generate URLs with base path
app.get('/api/info', (req, res) => {
    res.json({
        baseUrl: `${req.protocol}://${req.get('host')}${BASE_PATH}`,
        selfUrl: `${req.protocol}://${req.get('host')}${req.originalUrl}`
    });
});
```

Python FastAPI (Chatbot 04-05):

```
from fastapi import FastAPI, Request

app = FastAPI(
    title="Chatbot API",
    root_path="/chatbot04"  # Set base path
)
```

```python
@app.get("/info")
async def get_info(request: Request):
    return {
        "base_url": str(request.base_url),
        "path_info": request.url.path,
        "full_url": str(request.url)
    }
```

React Frontend Configuration:

```javascript
// Configure React Router with base path
import { BrowserRouter } from 'react-router-dom';

function App() {
    return (
        <BrowserRouter basename="/chatbot05">
            <Routes>
                <Route path="/" element={<Home />} />
                <Route path="/dashboard" element={<Dashboard />} />
            </Routes>
        </BrowserRouter>
    );
}

// Configure API base URL
const API_BASE = process.env.NODE_ENV === 'production'
    ? '/api/v2'
    : 'http://localhost:8002';
```

Challenge 6: Cookie and Session Management 🍪

The Problem:

Cookies have path and domain restrictions that can break with base path routing:

```javascript
// Backend sets cookie without considering base path
res.cookie('session', 'abc123', {
    path: '/'  // Available for entire domain
});

// Should be:
res.cookie('session', 'abc123', {
    path: '/chatbot01/'  // Only available for this app
});
```

Solution:

```
# Proxy cookie path rewriting
location /chatbot01/ {
    proxy_pass http://127.0.0.1:3000/;

    # Rewrite cookie paths in responses
    proxy_cookie_path / /chatbot01/;

    # Standard proxy headers
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
}
```

Challenge 7: WebSocket Connections 🔌

The Problem:

WebSocket connections for real-time features need special handling with base paths:

```
// Frontend WebSocket connection
const ws = new WebSocket('wss://project-1-xx.eduhk.hk/chatbot01/ws');
```

Solution:

```
location /chatbot01/ws {
    proxy_pass http://127.0.0.1:3000;

    # WebSocket specific headers
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;

    # Disable buffering for real-time
    proxy_buffering off;
    proxy_read_timeout 86400; # 24 hours
}
```

Testing and Debugging Base Path Issues 🔍

Comprehensive Testing Script:

```bash
#!/bin/bash
# Test all base path scenarios

DOMAIN="https://project-1-xx.eduhk.hk"

echo "Testing trailing slash redirects..."
for app in chatbot01 chatbot02 chatbot03 chatbot04 chatbot05; do
    echo "Testing /$app redirect:"
    curl -I "$DOMAIN/$app" | head -n 2
done

echo "Testing asset loading..."
curl -I "$DOMAIN/static/css/main.css"
curl -I "$DOMAIN/assets/logo.png"

echo "Testing API endpoints..."
curl -I "$DOMAIN/chatbot04/chat/completions"
curl -I "$DOMAIN/api/v2/health"

echo "Testing SPA routing..."
curl -s "$DOMAIN/chatbot05/dashboard" | grep -o "<title>.*</title>"
```

Nginx Debug Configuration:

```
# Temporary debugging location
location /debug {
    return 200 "
Debug Info:
Request URI: $request_uri
URI: $uri
Args: $args
Host: $host
Remote Addr: $remote_addr
Scheme: $scheme
Request Method: $request_method
";
    add_header Content-Type text/plain;
}
```

This comprehensive coverage of base path challenges and solutions should give you a thorough understanding of the complexities involved and how to address them systematically.

```
location /api/v1/ { proxy_pass http://app1:8001/; }
location /api/v2/ { proxy_pass http://app2:8002/; }
```

---

6. Reverse Proxy Fundamentals

What is a Reverse Proxy?

A reverse proxy sits between clients and backend servers, forwarding client requests to appropriate backends and returning responses to clients.

```
Client → Nginx (Reverse Proxy) → Backend Server
       ←                       ←
```

Forward Proxy vs Reverse Proxy

Aspect	Forward Proxy	Reverse Proxy
Position	Client-side	Server-side
Purpose	Hide client identity	Hide server details
Use Case	Corporate filtering	Load balancing, SSL termination
Example	Company proxy	Nginx, HAProxy

Essential Proxy Headers

Our Week 03 configuration uses these headers:

```
proxy_set_header Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Proto $scheme;
```

Header Explanations:

1. Host Header

```
proxy_set_header Host $host;
```

- **Purpose**: Preserve original host name
- **Value**: `project-1-xx.eduhk.hk`
- **Why needed**: Backend needs to know the original domain

2. X-Real-IP Header

```
proxy_set_header X-Real-IP $remote_addr;
```

- **Purpose**: Pass client's real IP address

- **Value**: `192.168.1.100` (example client IP)
- **Why needed**: Backend logs, security checks

3. X-Forwarded-For Header

```
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

- **Purpose**: Chain of proxy IPs
- **Value**: `192.168.1.100, 10.0.0.1` (client IP, proxy IPs)
- **Why needed**: Track request path through proxies

4. X-Forwarded-Proto Header

```
proxy_set_header X-Forwarded-Proto $scheme;
```

- **Purpose**: Original protocol (HTTP/HTTPS)
- **Value**: `https`
- **Why needed**: Backend needs to know if original request was secure

Week 03 Proxy Configurations Analysis

Standard Proxy (Chatbots 01-03)

```
location /chatbot01/ {
    proxy_pass http://127.0.0.1:3000/;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_redirect off;
}
```

Streaming Proxy (Chatbots 04-05 APIs)

```
location /chatbot04/chat/completions {
    proxy_pass http://127.0.0.1:8001/chat/completions;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_redirect off;

    # Streaming-specific configurations
    proxy_buffering off;          # Don't buffer responses
    proxy_cache off;              # Don't cache streaming responses
```

```
    proxy_set_header Connection '';    # Clear connection header
    proxy_http_version 1.1;       # HTTP/1.1 for persistent connections
    chunked_transfer_encoding off; # Handle chunked encoding
}
```

Proxy Path Handling

With Trailing Slash (Path Stripping)

```
location /chatbot01/ {
    proxy_pass http://127.0.0.1:3000/;
}
# Request: /chatbot01/api/health
# Proxied: http://127.0.0.1:3000/api/health
```

Without Trailing Slash (Path Preservation)

```
location /chatbot01/ {
    proxy_pass http://127.0.0.1:3000;
}
# Request: /chatbot01/api/health
# Proxied: http://127.0.0.1:3000/chatbot01/api/health
```

---

7. Static File Serving

Nginx as a Static File Server

Nginx excels at serving static files efficiently. It can handle thousands of concurrent requests for static content with minimal resource usage.

Root vs Alias Directive

Root Directive

```
location /images/ {
    root /var/www/html;
}
# Request: /images/photo.jpg
# File path: /var/www/html/images/photo.jpg
```

Alias Directive

```
location /images/ {
    alias /var/www/assets/;
}
# Request: /images/photo.jpg
# File path: /var/www/assets/photo.jpg
```

Week 03 Static File Examples

Chatbot 04: Mixed Static + Dynamic

```
# Serve static files directly
location /chatbot04/ {
    alias /home/proj07/project-1-xx/chatbot_04_MVPPython/;
    index index.html;
    try_files $uri $uri/ /chatbot04/index.html;
}

# Proxy API requests
location /chatbot04/chat/completions {
    proxy_pass http://127.0.0.1:8001/chat/completions;
}
```

File Structure:

```
/home/proj07/project-1-xx/chatbot_04_MVPPython/
├── index.html          # Main page
├── style.css           # Styles
├── script.js           # JavaScript
└── assets/            # Other static files
```

Request Handling:

- `GET /chatbot04/` → `index.html`
- `GET /chatbot04/style.css` → `style.css`
- `POST /chatbot04/chat/completions` → Python FastAPI backend

Chatbot 05: SPA with Asset Rewriting

```
# Handle React app asset paths
rewrite ^/assets/(.*)$ /chatbot05/assets/$1 last;

# Serve React build files
location /chatbot05/ {
    alias /var/www/html/chatbot05/;
    try_files $uri $uri/ /chatbot05/index.html;
```

```
    index index.html;
}

# Additional asset mapping
location /chatbot05/assets/ {
    alias /var/www/html/chatbot05/assets/;
}
```

Try Files Directive

The `try_files` directive attempts to serve files in order:

```
try_files $uri $uri/ /chatbot05/index.html;
```

Sequence:

1. Try to serve the exact URI as a file
2. Try to serve the URI as a directory (with index)
3. Fallback to `/chatbot05/index.html` (SPA fallback)

Example:

- Request: `/chatbot05/dashboard`
- Try 1: `/var/www/html/chatbot05/dashboard` (file) - Not found
- Try 2: `/var/www/html/chatbot05/dashboard/` (directory) - Not found
- Try 3: `/var/www/html/chatbot05/index.html` - Serve this (React handles routing)

Performance Optimizations

```
location ~* \.(css|js|png|jpg|jpeg|gif|ico|svg)$ {
    alias /var/www/html/assets/;
    expires 1y;                          # Cache for 1 year
    add_header Cache-Control "public, immutable";
    add_header Vary Accept-Encoding;     # Gzip consideration
    access_log off;                      # Don't log asset requests
}
```

---

8. Real-World Examples from Week 03

Let's analyze each chatbot configuration to understand different routing patterns:

Chatbot 01: Full-Stack Node.js Application

Architecture: Backend serves both API and frontend

```
```

```
location /chatbot01/ {
    proxy_pass http://127.0.0.1:3000/;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_redirect off;
}

location = /chatbot01 {
    return 301 /chatbot01/;
}
```

Backend Code (server.js):

```
const express = require('express');
const app = express();

// Serve static files from public directory
app.use(express.static('public'));

// API endpoints
app.post('/api/chat', (req, res) => { /* ... */ });
app.get('/api/health', (req, res) => { /* ... */ });

// Serve frontend for all other requests
app.get('*', (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'index.html'));
});
```

Request Flow:

1. Browser requests: `https://project-1-xx.eduhk.hk/chatbot01/`
2. Nginx matches: `location /chatbot01/`
3. Nginx proxies to: `http://127.0.0.1:3000/`
4. Node.js serves: `public/index.html`

Chatbot 02 & 03: Environment and CORS Demos

Same pattern as Chatbot 01, but different ports:

- Chatbot 02: Port 3001 (Environment Variables Demo)
- Chatbot 03: Port 3002 (CORS Configuration Demo)

Chatbot 04: Python FastAPI with Static Files

Architecture: Nginx serves static files, Python handles API

```
```

```
# Static file serving
location /chatbot04/ {
    alias /home/proj07/project-1-xx/chatbot_04_MVPPython/;
    index index.html;
    try_files $uri $uri/ /chatbot04/index.html;
}

# API endpoint
location /chatbot04/chat/completions {
    proxy_pass http://127.0.0.1:8001/chat/completions;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_redirect off;
    # Streaming configuration
    proxy_buffering off;
    proxy_cache off;
    proxy_set_header Connection '';
    proxy_http_version 1.1;
    chunked_transfer_encoding off;
}
```

Backend Code (main.py):

```
from fastapi import FastAPI

app = FastAPI()

@app.post("/chat/completions")
async def chat_completions(request: ChatRequest):
    # Handle API request
    return StreamingResponse(...)

@app.get("/health")
async def health():
    return {"status": "OK"}
```

Request Flow:

- `GET /chatbot04/` → Nginx serves `index.html`
- `GET /chatbot04/style.css` → Nginx serves static file
- `POST /chatbot04/chat/completions` → Python FastAPI backend

Chatbot 05: React SPA with Separate Backend

Architecture: React SPA + Python API with path rewriting

```
# Asset path rewriting for React build
rewrite ^/assets/(.*)$ /chatbot05/assets/$1 last;

# React SPA serving
location /chatbot05/ {
    alias /var/www/html/chatbot05/;
    try_files $uri $uri/ /chatbot05/index.html;
    index index.html;
}

# Asset directory mapping
location /chatbot05/assets/ {
    alias /var/www/html/chatbot05/assets/;
}

# API routes
location /api/v2/ {
    proxy_pass http://127.0.0.1:8002/;
    # ... proxy headers and streaming config
}
```

Request Flow Analysis:

1. **Initial Page Load**:
   - Request: `GET /chatbot05/`
   - Nginx serves: `/var/www/html/chatbot05/index.html`
2. **Asset Loading** (React Build Assets):
   - Request: `GET /assets/style.123abc.css`
   - Rewrite rule transforms to: `GET /chatbot05/assets/style.123abc.css`
   - Nginx serves: `/var/www/html/chatbot05/assets/style.123abc.css`
3. **Client-Side Routing**:
   - Request: `GET /chatbot05/dashboard` (React route)
   - try_files sequence:
     - Try: `/var/www/html/chatbot05/dashboard` (not found)
     - Try: `/var/www/html/chatbot05/dashboard/` (not found)
     - Serve: `/var/www/html/chatbot05/index.html` (React handles routing)
4. **API Calls**:
   - Request: `POST /api/v2/chat`
   - Nginx proxies to: `http://127.0.0.1:8002/chat`

Common Patterns Summary

Pattern	Use Case	Nginx Config	Backend Responsibility

Pattern	Use Case	Nginx Config	Backend Responsibility
Full Proxy	Traditional web apps	`proxy_pass` only	Serves everything
Static + API	Simple SPAs	`alias` + specific `proxy_pass`	API only
SPA + Rewrite	Complex React apps	`try_files` + `rewrite` rules	API + build optimization

9. Advanced Routing Patterns

URL Rewriting with Regular Expressions

Basic Rewrite Rules

```
# Rewrite assets for SPA
rewrite ^/assets/(.*)$ /chatbot05/assets/$1 last;

# Add trailing slash
rewrite ^/chatbot([0-9]+)$ /chatbot$1/ permanent;

# API versioning
rewrite ^/chatbot05/api/(.*)$ /api/v2/$1 last;
```

Rewrite Flags

- `last`: Stop processing, start new location lookup
- `break`: Stop processing in current location
- `redirect`: Return 302 temporary redirect
- `permanent`: Return 301 permanent redirect

Conditional Logic with Map Module

```
# Map user agent to backend
map $http_user_agent $backend {
    ~*mobile    mobile_backend;
    ~*tablet    tablet_backend;
    default     desktop_backend;
}

server {
    location /app/ {
        proxy_pass http://$backend;
    }
}
```

Load Balancing Between Multiple Backends

```
upstream chatbot_cluster {
    server 127.0.0.1:3000 weight=3;
    server 127.0.0.1:3001 weight=1;
    server 127.0.0.1:3002 backup;
}

location /chatbot/ {
    proxy_pass http://chatbot_cluster;
}
```

Rate Limiting

```
# Define rate limiting zone
limit_req_zone $binary_remote_addr zone=api:10m rate=10r/s;

location /api/ {
    limit_req zone=api burst=20 nodelay;
    proxy_pass http://backend;
}
```

Canary Deployments

```
# Route 10% of traffic to new version
map $arg_version $backend {
    v2      new_backend;
    default old_backend;
}

# Split based on IP hash
split_clients $remote_addr $backend {
    10%     new_backend;
    *       old_backend;
}

location /app/ {
    proxy_pass http://$backend;
}
```

---

10. Troubleshooting and Debugging

Common Nginx Routing Issues

1. 404 Not Found Errors

Problem: Request returns 404 even though file exists

Debugging Steps:

```
# Check nginx error log
sudo tail -f /var/log/nginx/error.log

# Test configuration
sudo nginx -t

# Check file permissions
ls -la /var/www/html/

# Check location matching
curl -I http://localhost/test-path
```

Common Causes:

- Wrong `alias` or `root` path
- File permissions (nginx user can't read)
- Location block not matching
- Missing `index` directive

2. 502 Bad Gateway Errors

Problem: Nginx can't reach backend service

Debugging:

```
# Check if backend is running
ps aux | grep node
ps aux | grep python

# Check if port is listening
sudo netstat -tlnp | grep :3000

# Test backend directly
curl http://localhost:3000/

# Check nginx error logs
sudo tail -f /var/log/nginx/error.log
```

3. Infinite Redirect Loops

Problem: Too many redirects error in browser

Common Cause:

```

```

```
# BAD: Creates redirect loop
location /chatbot01 {
    return 301 /chatbot01;  # Redirects to itself
}

# GOOD: Specific redirect
location = /chatbot01 {
    return 301 /chatbot01/; # Redirects to different path
}
```

4. Assets Not Loading (CORS/Path Issues)

Problem: CSS/JS files return 404 or wrong content-type

Debugging:

```
# Check request in browser dev tools
# Look for failed requests in Network tab

# Test asset loading directly
curl -I https://domain.com/assets/style.css

# Check location matching priority
nginx -T | grep -A 10 "location"
```

Debugging Tools and Techniques

1. Nginx Configuration Testing

```
# Test configuration syntax
sudo nginx -t

# Test and show configuration
sudo nginx -T

# Reload configuration (graceful)
sudo nginx -s reload
```

2. Request Tracing

```
# Add debug information to logs
access_log /var/log/nginx/debug.log combined;

# Custom log format for debugging
log_format debug '$remote_addr - $remote_user [$time_local] '
```

```
                    '"$request" $status $bytes_sent '
                    '"$http_referer" "$http_user_agent" '
                    'rt=$request_time upstream=$upstream_response_time';
```

3. Location Matching Test

```
# Test location to see which block matches
location /debug {
    return 200 "Matched /debug location\n";
    add_header Content-Type text/plain;
}
```

4. Variable Inspection

```
location /debug {
    return 200 "Host: $host
URI: $uri
Request URI: $request_uri
Args: $args
Remote Addr: $remote_addr
Scheme: $scheme
";
    add_header Content-Type text/plain;
}
```

Week 03 Specific Troubleshooting

Chatbot Applications Not Accessible

Checklist:

1. Backend service running on correct port?

```
ps aux | grep node    # For Node.js apps
ps aux | grep python  # For Python apps
```

2. Nginx configuration correct?

```
sudo nginx -t
sudo nginx -s reload
```

3. Firewall allowing traffic?

```
sudo ufw status
sudo ufw allow 80
sudo ufw allow 443
```

4. SSL certificate valid?

```
openssl x509 -in /etc/nginx/ssl/cert.crt -text -noout
```

Asset Loading Issues for SPAs

Problem: React app loads but assets (CSS/JS) return 404

Solution Check:

```
# Ensure rewrite rule is correct
rewrite ^/assets/(.*)$ /chatbot05/assets/$1 last;

# Verify asset directory exists
ls -la /var/www/html/chatbot05/assets/

# Check file permissions
sudo chown -R www-data:www-data /var/www/html/chatbot05/
```

---

11. Performance and Security Considerations

Performance Optimization

1. Static File Caching

```
location ~* \.(css|js|png|jpg|jpeg|gif|ico|svg|woff|woff2)$ {
    expires 1y;
    add_header Cache-Control "public, immutable";
    add_header Vary Accept-Encoding;
    access_log off;
}
```

2. Gzip Compression

```
gzip on;
gzip_vary on;
gzip_min_length 1024;
gzip_types text/plain text/css text/xml text/javascript
```

```
            application/x-javascript application/xml+rss
            application/javascript application/json;
```

3. Connection Keep-Alive

```
upstream backend {
    server 127.0.0.1:3000;
    keepalive 32;
}

location /api/ {
    proxy_pass http://backend;
    proxy_http_version 1.1;
    proxy_set_header Connection "";
}
```

4. Worker Process Optimization

```
worker_processes auto;
worker_connections 1024;
worker_rlimit_nofile 2048;
```

Security Best Practices

1. Hide Nginx Version

```
server_tokens off;
```

2. Security Headers

```
add_header X-Frame-Options "SAMEORIGIN" always;
add_header X-XSS-Protection "1; mode=block" always;
add_header X-Content-Type-Options "nosniff" always;
add_header Referrer-Policy "no-referrer-when-downgrade" always;
add_header Content-Security-Policy "default-src 'self' http: https: data: blob:
'unsafe-inline'" always;
```

3. SSL Security (From Week 03)

```
```

```
ssl_protocols TLSv1.2 TLSv1.3;
ssl_prefer_server_ciphers on;
ssl_ciphers HIGH:!aNULL:!MD5;
ssl_stapling on;
ssl_stapling_verify on;
```

4. Rate Limiting

```
limit_req_zone $binary_remote_addr zone=general:10m rate=10r/s;
limit_req_zone $binary_remote_addr zone=api:10m rate=5r/s;

location / {
    limit_req zone=general burst=20 nodelay;
}

location /api/ {
    limit_req zone=api burst=10 nodelay;
}
```

5. Access Control

```
# Block specific user agents
if ($http_user_agent ~ (bot|crawler|spider)) {
    return 403;
}

# Allow only specific IPs for admin
location /admin/ {
    allow 192.168.1.0/24;
    allow 10.0.0.0/8;
    deny all;
}
```

---

12. Practical Exercises

Exercise 1: Basic Location Matching

Create nginx configuration that handles these requirements:

1. Redirect `https://domain.com/app` to `https://domain.com/app/`
2. Serve static files from `/var/www/app/` for paths starting with `/app/`
3. Proxy API requests from `/app/api/` to `http://localhost:3000/api/`

▶ Solution

```
server {
    listen 443 ssl;
    server_name domain.com;

    # SSL configuration here...

    # Redirect without trailing slash
    location = /app {
        return 301 /app/;
    }

    # API proxy (more specific, matches first)
    location /app/api/ {
        proxy_pass http://localhost:3000/api/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # Static files (less specific, matches after API)
    location /app/ {
        alias /var/www/app/;
        index index.html;
        try_files $uri $uri/ /app/index.html;
    }
}
```

Exercise 2: Multi-App Routing

Design nginx configuration for three applications:

- Blog app at `/blog/` (Node.js on port 3001)
- Shop app at `/shop/` (Python on port 8001)
- Admin app at `/admin/` (React SPA + API at port 8002)

Requirements:

- All apps should have trailing slash redirects
- Admin app needs API routes at `/admin/api/`
- Shop app needs static file serving

▶ Solution

```
server {
    listen 443 ssl;
    server_name mysite.com;

    # Trailing slash redirects
    location = /blog { return 301 /blog/; }
```

```
    location = /shop { return 301 /shop/; }
    location = /admin { return 301 /admin/; }

    # Blog app (full proxy)
    location /blog/ {
        proxy_pass http://127.0.0.1:3001/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # Shop app (static files + API)
    location /shop/api/ {
        proxy_pass http://127.0.0.1:8001/api/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /shop/ {
        alias /var/www/shop/;
        index index.html;
        try_files $uri $uri/ /shop/index.html;
    }

    # Admin app (React SPA + API)
    location /admin/api/ {
        proxy_pass http://127.0.0.1:8002/api/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /admin/ {
        alias /var/www/admin/;
        try_files $uri $uri/ /admin/index.html;
        index index.html;
    }
}
```

Exercise 3: Debugging Scenario

You have this configuration but users report that CSS files are not loading for the SPA:

```
location /myapp/ {
    alias /var/www/myapp/build/;
    try_files $uri $uri/ /myapp/index.html;
}
```

The React app is built and tries to load `/static/css/main.css`, but gets 404.

Task: Identify the problem and provide a solution.

▶ Analysis and Solution

Problem: React build assumes root path for assets. The app loads from `/myapp/` but assets are requested from `/static/css/main.css` (root path).

Solutions:

Option 1 - Add rewrite rule:

```
# Rewrite root assets to app assets
rewrite ^/static/(.*)$ /myapp/static/$1 last;

location /myapp/ {
    alias /var/www/myapp/build/;
    try_files $uri $uri/ /myapp/index.html;
}
```

Option 2 - Add specific location for assets:

```
location /myapp/ {
    alias /var/www/myapp/build/;
    try_files $uri $uri/ /myapp/index.html;
}

location /static/ {
    alias /var/www/myapp/build/static/;
}
```

Option 3 - Configure React build with PUBLIC_URL:

```
# In React build process
PUBLIC_URL=/myapp npm run build
```

Exercise 4: Performance Optimization

Optimize this basic configuration for a high-traffic site:

```
server {
    listen 443 ssl;
    server_name example.com;
```

```
    location /api/ {
        proxy_pass http://backend;
    }

    location / {
        root /var/www/html;
    }
}
```

Add caching, compression, and security headers.

▶ Optimized Solution

```
# Add upstream with keepalive
upstream backend {
    server 127.0.0.1:8000;
    keepalive 32;
}

server {
    listen 443 ssl http2;  # Enable HTTP/2
    server_name example.com;

    # Security headers
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-XSS-Protection "1; mode=block" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header Strict-Transport-Security "max-age=63072000" always;

    # Gzip compression
    gzip on;
    gzip_vary on;
    gzip_min_length 1024;
    gzip_types text/plain text/css application/json application/javascript
text/xml application/xml application/xml+rss text/javascript;

    # API with keepalive
    location /api/ {
        proxy_pass http://backend;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # API caching for GET requests
        proxy_cache_methods GET HEAD;
        proxy_cache_valid 200 5m;
    }

    # Static files with aggressive caching
```

```
    location ~* \.(css|js|png|jpg|jpeg|gif|ico|svg|woff|woff2)$ {
        root /var/www/html;
        expires 1y;
        add_header Cache-Control "public, immutable";
        add_header Vary Accept-Encoding;
        access_log off;
    }

    # HTML files with shorter cache
    location ~* \.html$ {
        root /var/www/html;
        expires 1h;
        add_header Cache-Control "public, must-revalidate";
    }

    # Default location
    location / {
        root /var/www/html;
        index index.html;
        try_files $uri $uri/ /index.html;
    }
}
```

Conclusion

This tutorial covered the fundamental concepts of nginx and base path routing using real-world examples from our Week 03 multi-chatbot architecture. Key takeaways:

Core Concepts Mastered:

1. **Server Blocks**: Virtual host configuration and request matching
2. **Location Directives**: URL pattern matching and priority rules
3. **Base Path Routing**: Multi-application hosting under single domain
4. **Reverse Proxy**: Request forwarding and header management
5. **Static File Serving**: Efficient content delivery and caching

Architecture Patterns Learned:

- **Full Proxy**: Backend handles everything (Chatbots 01-03)
- **Hybrid Static+API**: Nginx serves files, backend handles API (Chatbot 04)
- **SPA with Rewriting**: Complex React apps with asset path resolution (Chatbot 05)

Best Practices Applied:

- Trailing slash consistency
- Proper proxy headers for backend integration
- SSL/TLS security configuration
- Performance optimization techniques
- Debugging and troubleshooting methodologies

Next Steps:

- Implement monitoring and logging
- Add advanced features like load balancing
- Explore containerization with Docker
- Study advanced nginx modules and plugins

This foundation enables you to design, implement, and maintain robust nginx configurations for complex multi-application environments.