# CS 447/647

Web Hosting

What are the two most popular open-source HTTP servers?

What are the components of a web application? (HTTP, UWSGI, Rev-Proxy...)

What is a URL?

What are the HTTP verbs (transactions)?

What are common HTTP responses?

What are the parts of a HTTP request?

# Web Hosting

- Linux and *NIX are the dominant platforms for serving WWW
  - 67% of the top one millions sites
- Open source is 80% of the web server software market
- Web applications are a set of components
  - Static content
  - Dynamic content
  - Databases
- Cloud infrastructure
  - Provision quickly
  - Simplifies design and deployment

# HyperText Transfer Protocol

- Core application level protocol for the web
  - Essential for all Sysadmins
    - DevOps, SWEs, SREs too
- Client/Server Architecture
  - One Request - One Response
  - Typical web page needs multiple requests
- Adapts slowly
  - It’s popularity makes changes high stakes
  - Wasn’t always this way…
    - https://thehistoryoftheweb.com/the-origin-of-the-img-tag/

# Hypertext Transfer Protocol

- HTTP Versions 1.0 & 1.1
  - Plain text protocol
    - telnet google.com 80
    - GET /index.html
- HTTP/2 under development
  - Backwards compatible
  - Performance Improvements
  - Only TLS-encrypted
  - Binary Format

# Uniform Resource Locators (URLs)

- Defines how and where to access a resource
- Not unique to HTTP
  - FTP, SMB, git ssh
- Pattern for URL is scheme:address
  - Scheme is a protocol
  - Address is the resource
  - mailto:newellz2@unr.edu
  - smb://storage.unr.edu
    - Microsoft: \\storage.unr.edu\

# URLs

Address Structure

scheme://[username:password@]hostname[:port][/path][?query][#anchor]

- Username and password are HTTP Basic Authentication
  - Bad idea because URLs are logged
  - Not encrypted by default
- Hostname is a domain name or IP
- Port is the TCP port. Defaults 80 and 443
- Query is a set of key value pairs
  - key=value
- Anchor is a subtarget of a URL.
  - <h1 id=”Intro>Introduction </h1>

# HTTP Transactions

- HTTP requests
  - First line is the action then path

```
GET /index.php HTTP/1.1
```

| Verb | Safe? | Purpose |
|---|---|---|
| GET | Yes | Retrieves the specified resource |
| HEAD | Yes | Like GET, but requests no payload; retrieves metadata only |
| DELETE | No | Deletes the specified resource |
| POST | No | Applies request data to the given resource |
| PUT | No | Similar to POST, but implies replacement of existing contents |
| OPTIONS | Yes | Shows what methods the server supports for the specified path |

# HTTP request methods

- GET - Most common verb followed by POST
- POST vs PUT
  - PUT should be idempotent
  - Server that send mails should use POST
  - PUT used for APIs
- DELETE
  - Often used with APIs
- OPTIONS
  - Bots

https://couchdb.apache.org/

# HTTP Responses

| Code | General indication | Examples |
|---|---|---|
| 1xx | Request received; processing continues | 101 Switching Protocols |
| 2xx | Success | 200 OK<br>201 Created |
| 3xx | Further action needed | 301 Moved Permanently<br>302 Found [a] |
| 4xx | Unsatisfiable request | 403 Forbidden<br>404 Not Found |
| 5xx | Server or environment failure | 503 Service Unavailable |

a. Most often used (inappropriately, according to the spec) for temporary redirects

# 418 I'm a teapot

“Any attempt to brew coffee with a teapot should result in the error code "418 I'm a teapot". The resulting entity body MAY be short and stout.”

Hyper Text Coffee Pot Control Protocol (HTCPCP/1.0)

# Headers and Message Body

| Name: example value | Dir[a] | Content |
|---|---|---|
| Host: www.admin.com | → | Domain name and port being requested |
| Content-Type: application/json | ↔ | Data format wanted or contained |
| Authorization: Basic QWx...FtZ== | → | Credentials for HTTP basic authentication |
| Last-Modified: Wed, Sep 7 2016... | ← | Object's last known modification date |
| Cookie: flavor=oatmeal | → | Cookie returned from a user agent |
| Content-Length: 423 | ↔ | Length of the body in bytes |
| Set-Cookie: flavor=oatmeal | ← | Cookie to be stored by the user agent |
| User-Agent: curl/7.37.1 | → | User agent submitting the request |
| Server: nginx/1.6.2 | ← | Server software responding to the request |
| Upgrade: HTTP/2.0 | ↔ | Request to change to another protocol |
| Expires: Sat, 15 Oct 2016 14:02:... | ← | Length of time the response can be cached |
| Cache-Control: max-age=7200 | ↔ | Like Expires, but allows more control |

a. Direction: → request-only, ← response-only, or ↔ both

# Headers

- Headers are separate from the body
- Body can be text, binary or blank

curl -i \
**-H "Accept: application/json" **
**-H "Content-Type: application/json" **
http://host/resource

# curl - HTTP from the command line

```
$ curl -s -v -o /dev/null http://admin.com
* Rebuilt URL to: http://admin.com/
* Hostname was NOT found in DNS cache
*   Trying 54.84.253.153...
* Connected to admin.com (54.84.253.153) port 80 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.37.1
> Host: admin.com
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Mon, 27 Apr 2015 18:17:08 GMT
* Server Apache/2.4.7 (Ubuntu) is not blacklisted
< Server: Apache/2.4.7 (Ubuntu)
< Last-Modified: Sat, 02 Feb 2013 03:08:20 GMT
< ETag: "66d3-4d4b52c0c1100"
< Accept-Ranges: bytes
< Content-Length: 26323
< Vary: Accept-Encoding
< Content-Type: text/html
<
{ [2642 bytes data]
* Connection #0 to host admin.com left intact
```

# curl

- libcurl
  - client-side URL transfer library
  - DICT, FILE, FTP, FTPS, Gopher, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMTP, SMTPS, Telnet and TFTP

```
CURL *curl;
CURLcode res;

curl_global_init(CURL_GLOBAL_DEFAULT);

curl = curl_easy_init();
if(curl) {
  curl_easy_setopt(curl, CURLOPT_URL, "https://example.com/");
```

# curl Examples

```
curl https://www.google.com/accounts/ClientLogin \
--data-urlencode Email=brad.gushue@example.com --data-urlencode
Passwd=new+foundland \
-d accountType=GOOGLE \
-d source=Google-cURL-Example \
-d service=lh2
```

```
curl -X DELETE http://couch.db/index/resource
```

# TCP connection reuse

- TCP connections are expensive
  - 3-way handshake before request
- Average site has 99 resources per page
  - New connection for each would be slow
- “Connection: Keep-Alive” header allows reuse
  - Multiple requests over one connections
  - Non-trivial performance hit
    - Servers must maintain thousands of connections
- HTTP/2 introduces multiplexing
  - Multiple transactions on a single connection
  - Lower overhead

# HTTP over TLS

- HTTP Provides no network-level security
  - URLs, Headers and payload are open
- Transport Layer Security runs between HTTP and TCP
  - Encryption and authentication
- TLS replaced Secure Socket Layer (SSL)
  - SSL deprecated
  - Assume SSL means TLS
- TLS verifies the server’s identity (domain name)
  - Certificate Authorities (ca-certificates)

# Virtual hosts

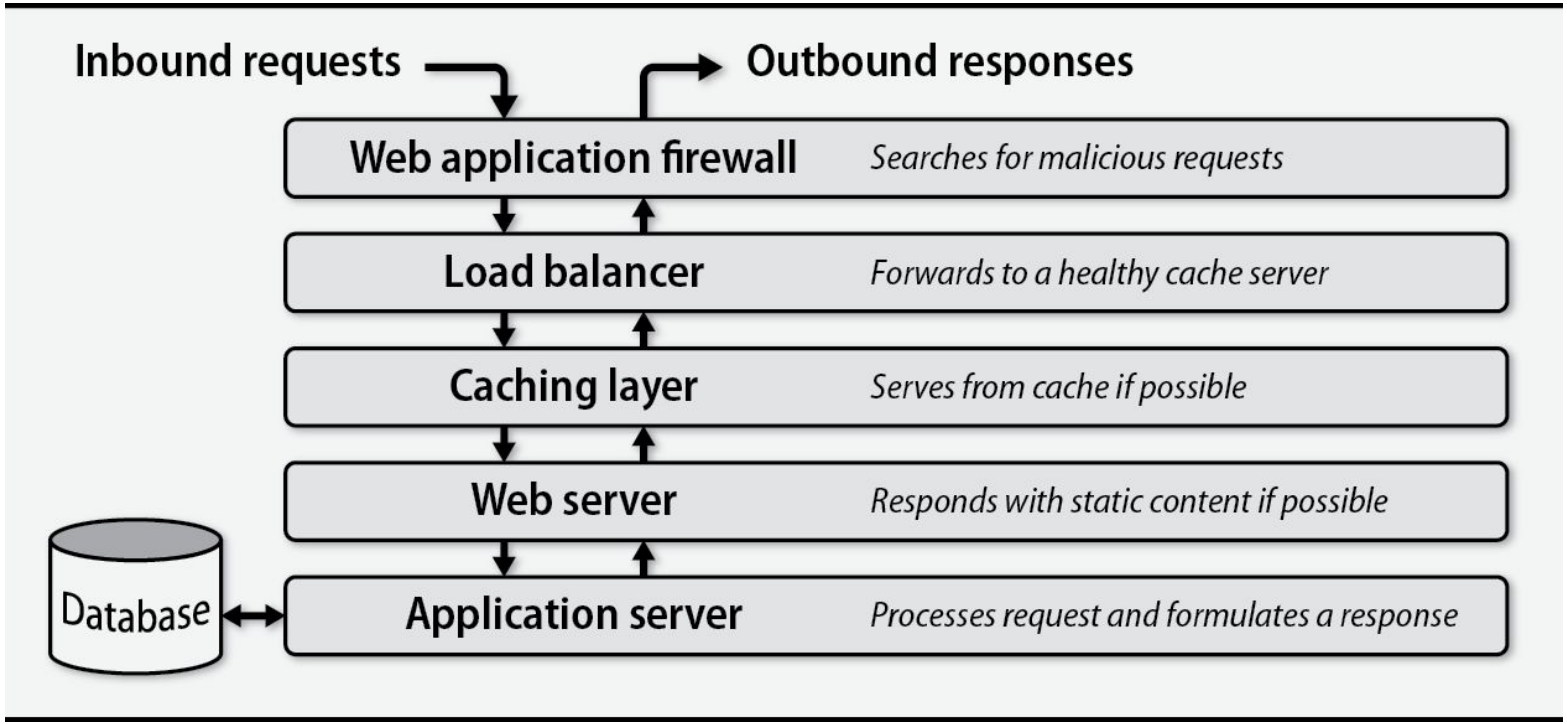
- Many domain names, one server
  - A records
  - CNAME records
- RFC2616 - Host header
  - Server examines the header
  - Required in HTTP 1.1
- Issues with TLS
  - TLS session required before Host header submitted
  - Server Name Indication (SNI) adds hostname to initial TLS message

# Web Software

| Type | Purpose | Examples |
|---|---|---|
| Application server | Runs web app code, interfaces to web servers | Unicorn, Tomcat |
| Cache | Speeds access to frequently requested content | Varnish, Squid |
| Load balancer | Relays requests to downstream systems | Pound, HAProxy |
| Web app firewall [a] | Inspects HTTP traffic for common attacks | ModSecurity |
| Web server | Serves static content, couples to other servers | Apache, NGINX |

a. Often abbreviated WAF

# Web Software



# Web Servers and Proxies

- Web Server Content
  - Static content
  - Dynamic content
- Web Server Features
  - Virtual Hosts
  - Handling TLS
  - Logging requests and responses
  - HTTP basic authentication
  - Routing to downstream systems
  - Execution of Dynamic content

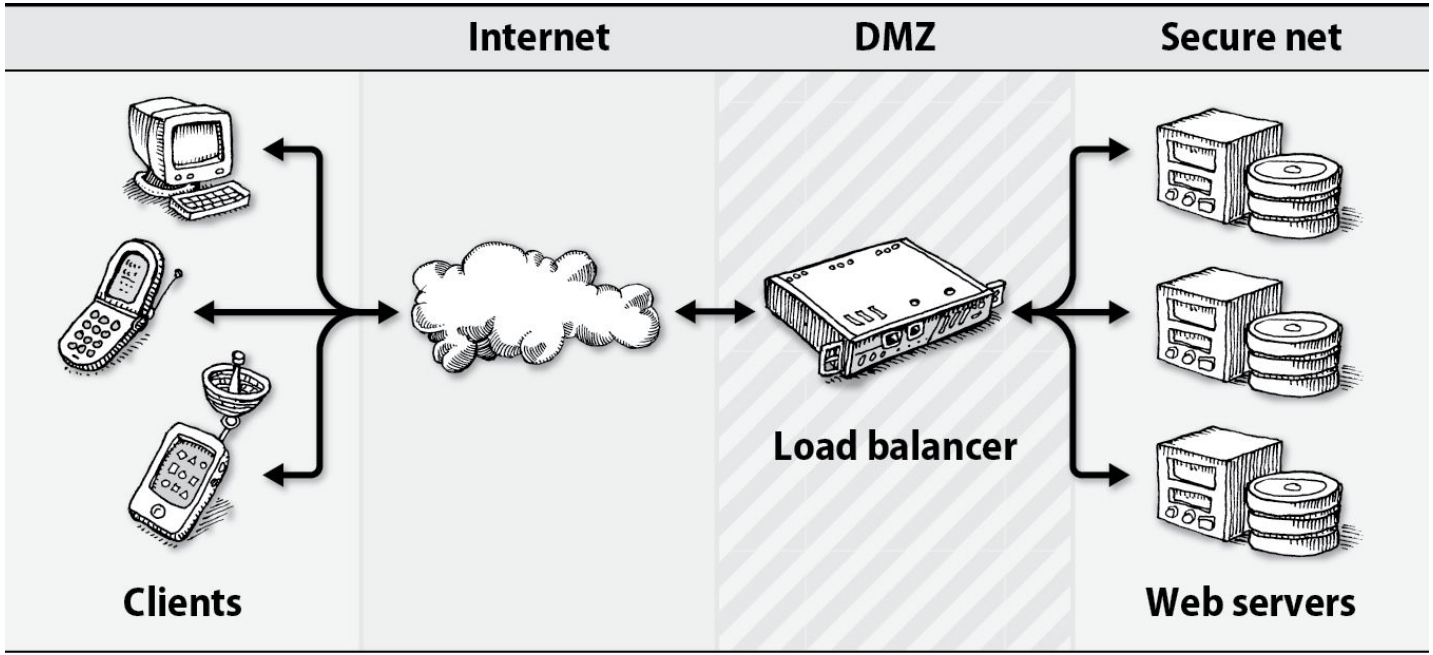
# Web Servers

- Popular servers
  - NGINX
    - 20% of websites
    - Fairly new
    - High-performance, event-based
  - apache2
    - 46 % of websites
    - Released in 1995
    - Reference HTTP implementation
    - Needs significant tweaking to get high performance
      - Multi-Processing Module pre-forking
  - IIS
    - Windows Web Server

# Load balancers

- Route requests to other systems
  - Handle lots of concurrent requests
- Allows you to drain resources
  - Easily remove a server from rotation
- Health check mechanism can remove a server from rotation
  - HTTP 50x responses
- Typically run in pairs
  - Passive backup
  - Virtual IP
- Load Balancing algorithms
  - Round robin - distributed in fixed order across servers
  - Load equalization - new request go to server with least connections
  - Partitioning - Server selected based on a client's request

# Load balancers



# Load balancers

- TLS Termination
  - Backend WWW server won’t need HTTPs
- Distribute other traffic
  - SQL
- Common load balancers
  - nginx
  - HAProxy
  - F5
  - Amazon Elastic Load Balancer (ELB)

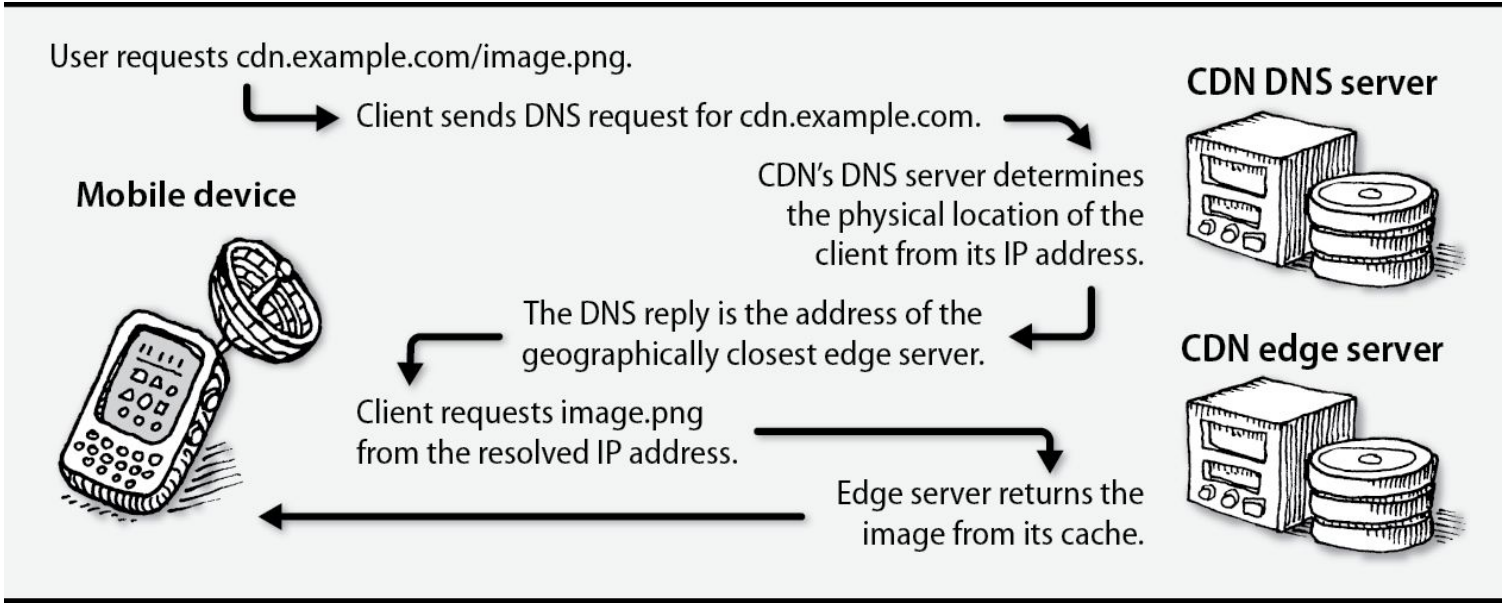
# Caches

- Clients frequently access the same content
  - CSS, Images, SQL queries
- Adds complexity
  - Stale content
  - curl -H "Cache-Control: no-cache"

| Server | Notes |
| --- | --- |
| Squid | One of the first open source cache implementations<br>Normally used as a proxy cache<br>Includes important features like antivirus and TLS |
| Varnish | Exceptional configuration language<br>Multithreaded<br>Modular and extensible |
| Apache mod_cache | Good choice for sites already running **httpd** |
| NGINX | Good choice for sites already running NGINX<br>Has a reputation for good performance |
| Apache Traffic Server | Runs at extremely high-traffic sites<br>Supports HTTP/2<br>Donated to the Apache Foundation by Yahoo! |

# Content Delivery Networks

- Global distributed system for content
  - Improves web performance
    - Content is closer to users
    - Youtube
- Traditionally for static content
  - Images, CSS and javascript
- Requests to a CDN are routed to closest server
  - Edge servers are like proxy caches


User requests cdn.example.com/image.png.
Client sends DNS request for cdn.example.com.
CDN DNS server
CDN's DNS server determines the physical location of the client from its IP address.
Mobile device
The DNS reply is the address of the geographically closest edge server.
CDN edge server
Client requests image.png from the resolved IP address.
Edge server returns the image from its cache.

# Content Delivery Networks

Examples:

```
<script src="https://ajax.googleapis.com/ajax/libs/d3js/5.15.0/d3.min.js"></script>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>

<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
crossorigin="anonymous">
```

# Languages of the web

- Ruby - Ruby-on-Rails
- **Python** - Django, **Flask**
- Java - Grails
- Node.js - Server-side Javascript
- PHP - Personal Home Page, Wordpress, Drupal, CakePHP
- **Go** - Google’s lower-level language
- .NET - Microsoft’s framework, ASP.NET MVC, C#

# Flask

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'
```

# Go

```
package main

import (
    "fmt"
    "log"
    "net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hi there, I love %s!", r.URL.Path[1:])
}

func main() {
    http.HandleFunc("/", handler)
    log.Fatal(http.ListenAndServe(":8080", nil))
}
```

# Application Programming Interfaces (APIs)

- Intended for software agents
  - python3-requests
- Exposed to the world
  - Promote use by eternal developers
- Use normal HTTP Requests
- Text-based Serialization
  - JSON - Can’t be used for evil
  - XML

# API Example

```
$ curl https://api.spotify.com/v1/artists/3WrFJ7ztbogyGnTHbHJFl2 | jq '.'
{
  "external_urls": {
    "spotify": "https://open.spotify.com/artist/3WrFJ7ztbogyGnTHbHJFl2"
  },
  "followers": {
    "href": null,
    "total": 1566620
  },
  "genres": [ "british invasion" ],
  "href": "https://api.spotify.com/v1/artists/3WrFJ7ztbogyGnTHbHJFl2",
  "id": "3WrFJ7ztbogyGnTHbHJFl2",
  "images": [ <removed for concision> ],
  "name": "The Beatles",
  "popularity": 91,
  "type": "artist",
  "uri": "spotify:artist:3WrFJ7ztbogyGnTHbHJFl2"
}
```

# API Types

- REST - Representational State Transfer
  - Typically JSON
  - Canvas - `/api/v1/users/:user_id/courses`
- SOAP - “Simple” Object Access Protocol
  - Complex XML based format
  - Few URLs, Large Requests, Poor Performance
  - Microsoft..

# Web hosting in the cloud

- Lots of providers
  - Google, Amazon, Microsoft, Linode
  - Platform as a service - AppEngine
  - VMS - Linode
- Build vs Buy
  - Raw VMS
  - Hybrid - AWS Load-Balancer with VMS’
  - Avoid DIY unless it’s a core competency
- Static Sites
  - Google Firebase - Copy and Paste

# nginx

- Install
  - `apt install nginx-extras`
  - extras = Standard HTTP, Optional, Mail and 3rd Party
    - FastCGI, SSL, IMAP, PAM
- Uses signals for some housekeeping

| Signal | Function |
|---|---|
| TERM or INT | Shuts down immediately |
| QUIT | Completes and closes all current connections, then shuts down |
| USR1 | Reopens log files (used to facilitate log rotations) |
| HUP | Reloads the configuration[a] |
| USR2 | Gracefully replaces the server binary without interrupting service[b] |

a. This option tests the syntax of the new configuration, and if the syntax is valid, starts new workers with the new configuration. It then gracefully shuts down the old workers.

b. See the **nginx** command-line documentation for details on how this works.

# Configuring nginx

| | RHEL/CentOS | Debian/Ubuntu | FreeBSD |
|---|---|---|---|
| Package name | nginx [a] | nginx | nginx |
| Daemon path | **/sbin/nginx** | **/usr/sbin/nginx** | **/usr/local/sbin/nginx** |
| Configuration root | **/etc/nginx** | **/etc/nginx** | **/usr/local/etc/nginx** |
| Virtual host config [b] | **conf.d/** | **sites-available/ sites-enabled/** | *No prescribed location* |
| Default user | nginx | www-data | nobody |

a. You must enable the EPEL software repository; see fedoraproject.org/wiki/EPEL.
b. Relative to the configuration root directory

# Configuring nginx

Contexts {}

```
events { }

http {
    server {
        server_name www.admin.com;
        root /var/www/admin.com;
    }
}
```

# Configuring nginx

```
http {
    server {
        server_name admin.com www.admin.com;
        root /var/www/admin.com;
    }
    server {
        server_name example.com www.example.com;
        root /var/www/example.com;
    }
}
```

# Configuring TLS

```
server {
    listen 443;
    ssl on;
    ssl_certificate /etc/ssl/certs/admin.com.crt;
    ssl_certificate_key /etc/ssl/private/admin.com.crt;
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE... # truncated
    ssl_prefer_server_ciphers on;

    server_name admin.com www.admin.com;
    root /var/www/admin.com/site1;
}
```

# Configuring headers

```
location / {
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-Host $server_name;
set_by_lua $client_cert "return ngx.var.ssl_client_raw_cert
                     and ngx.var.ssl_client_raw_cert:gsub('\\n','
') or nil";
uwsgi_param X-SSL-CERT $ssl_client_raw_cert;

}
```

```
upstream admin-servers {
    server web1.admin.com:8080 max_fails=2;
    server web2.admin.com:8080 max_fails=2;
}

http {
    server {
        server_name admin.com www.admin.com;
        location / {
            proxy_pass http://admin-servers;
            health_check interval=30 fails=3 passes=1 uri=/health_check
                match=admin-health  # line not split in original file
        }
    }
    match admin-health {
        status 200;
        header Content-Type = text/html;
        body ~ "Red Leader, Standing By";
    }
}
```

# Reverse Proxying

- Nginx serves static content and load balances
- Web Server Gateway Interface (WSGI) handles dynamic content
  - Python, Ruby, Perl