NGINX CORE

Flawless Application Delivery

Prerequisites/Expectations

- Sysadmin, DevOps, Solution Architect
- Some familiarity with Web Servers
- Some familiarity with Linux
- Text Editor: Vim, Vi, Emacs etc.
- Some knowledge of Networking

The Training Environment

- AWS EC2 Instances
- Ubuntu
- NGINX Plus

Log Into AWS

If you haven't done so already, please take the time to SSH into your EC2 Instances (Windows users use PuTTY).

Check your email for the login credentials, check your spam folder!

```
ssh student<number>@<ec2-server-hostname>
```

Course Administration

- Course Duration: 7-8 hours
- 10 minute break at the top of each hour
- 30-60 min break after part 1
- Ask questions at any time!

Agenda: Part One

- Overview
- Serving Static Content
- Proxying Connections
- Logging
- Security
- Variables

NGINX OVERVIEW

Module Objectives

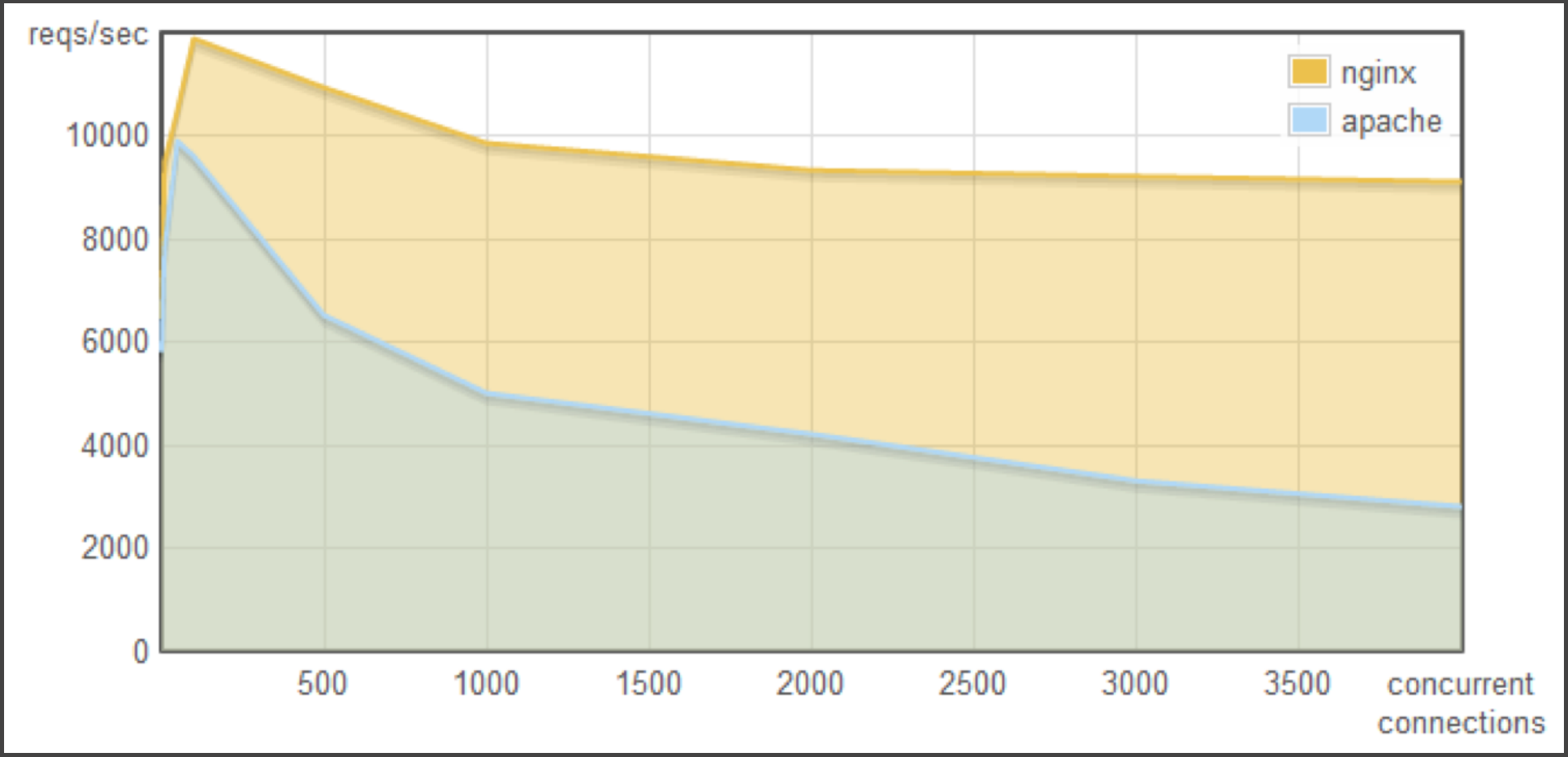
This module enables you to:

- Gain a basic understanding of NGINX's features
- Learn about the history of NGINX
- Understand the various use cases that NGINX supports

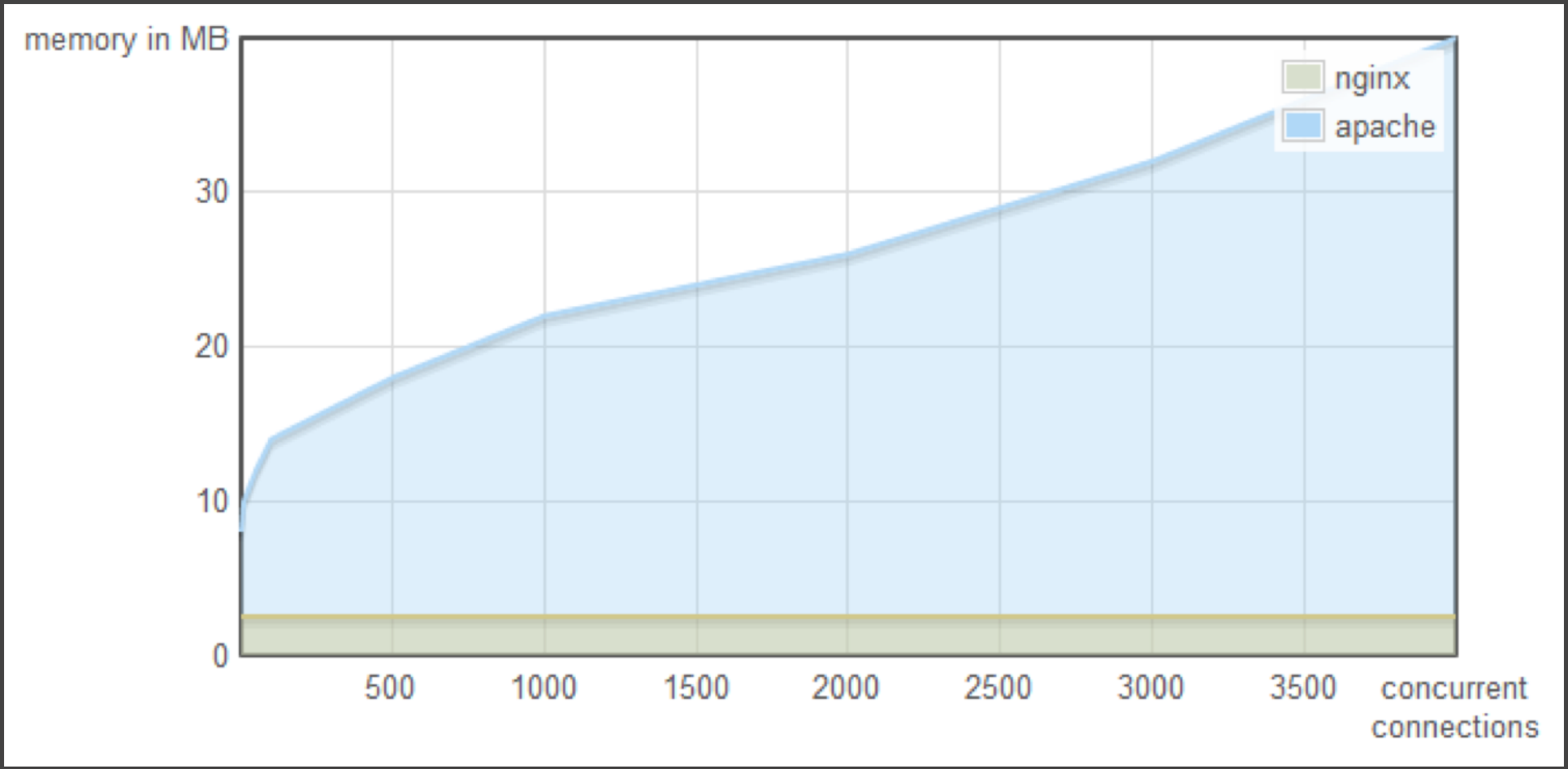
Origins of NGINX

- 2002: Igor Sysov working for rambler.ru
- 2004: First OSS release
- 2011: Company founded
- 2016-Present: 500+ customers and 80+ employees

High Concurrency

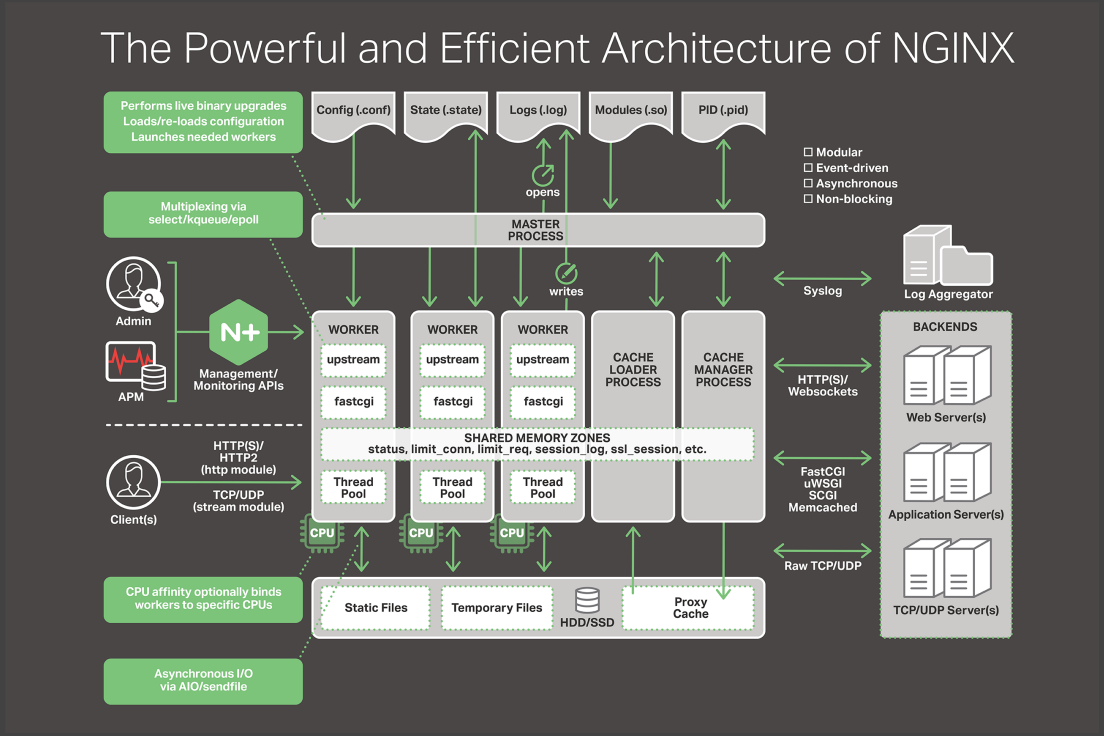


Low Memory


memory in MB
nginx
apache
30
20
10
0
500
1000
1500
2000
2500
3000
3500
concurrent
connections

NGINX Use Cases

NGINX Architecture



`Master:` Evaluates config, modules, PID, logs

`Workers:` Handles requests and responses

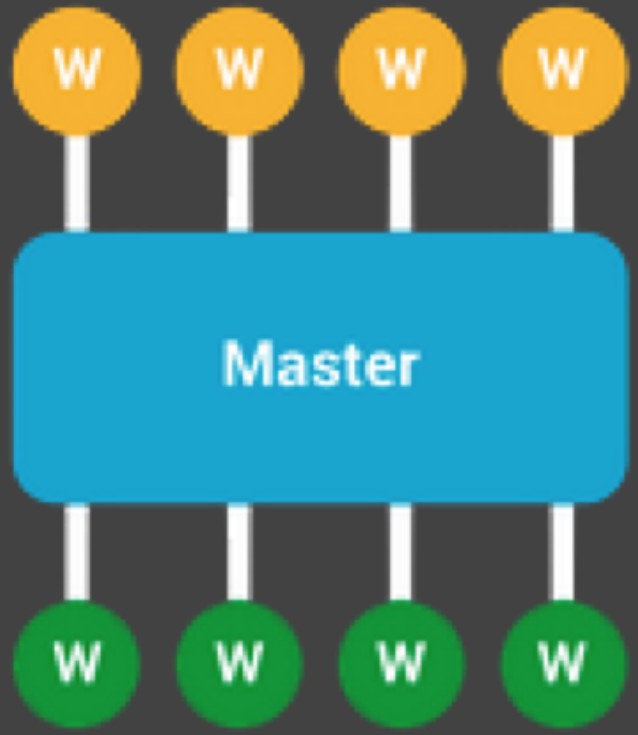
`Shared Memory:` Counters rate limits, status, session log etc.

Basic NGINX Commands

```
#reloads config
$ nginx -s reload

#graceful shutdown
$ nginx -s quit

#terminates nginx process
$ nginx -s stop

#config syntax check (pre-reload)
$ nginx -t

#displays currently running configs
$ nginx -T

#checks nginx version
$ nginx -v
```

Reloading the Configuration

1. `nginx -s reload` sends `SIGHUP` signal to kernel
2. Master evaluates new config
3. Checks for `emerg` level errors
4. Forks new workers while old workers gracefully shut down



SERVING STATIC CONTENT

Module Objectives

This module enables you to:

- Understand the Configuration File
- Configure a Basic Setup
- Explore server selection methods

Running Processes

To check running processes, run the following command:

```
$ ps aux | grep nginx
```

Path to Files

Executable Path

```
$ /usr/sbin/nginx
```

Log Path

```
$ /var/log/nginx
```

Configuration File

Global Configuration Path

```
$ /etc/nginx/nginx.conf
```

Additional Configuration(s) Path

```
$ /etc/nginx/conf.d/*.conf
```

Documentation: nginx.conf Example

Include Directive

The following line in nginx.conf allows NGINX to search for additional configurations

```
include /etc/nginx/conf.d/*.conf;
```

Configuration File Structure

- Directives
- Blocks
- Contexts

Documentation: How .conf Files Work

Directives

Configuration statement that controls NGINX Modules

```
listen 80;
root /usr/share/nginx/html;
index index.html index.htm index.php;
```

Blocks

Contains mixture of directives and data—begins and ends with curly brackets.

```
server {
      listen 80;
      root /usr/share/nginx/html;
      index index.html index.htm index.php;
  }
```

Contexts

Nested Blocks implying a hierarchy. Colloquially, 'Block' and 'Context' are interchangeable.

```
http {
        include       /etc/nginx/mime.types;
        default_type  application/octet-stream;
        gzip on;
        log_format

        server {
                listen 80;
                root /usr/share/nginx/html;
                index index.html index.htm index.php;

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

Documentation: Further Explanation

Serving Content

Requirements:

- `http` - high level processing (logging, compression, caching etc.)
- `server` - virtual server that handles the request
- `location` - processing based on request URI

`server` Block Example

- Defines virtual `server` ("VirtualHost" in Apache)
- Always nested inside either `http` or `stream` context
- Binds to TCP sockets with `server_name` and `listen`

```
server {
    listen 80;
    server_name www.example.com;
    root /etc/student1/public_html;
}
```

Documentation: Server Block Examples

`listen` Directive

- Defines IP / Port that `server` responds to
- Default is `0.0.0.0:80` (`:8080` for non-root)
- Can be: IP, IP:Port, Port, Unix Socket

`listen` Rules

NGINX will substitute values for "incomplete" directives

- No `listen` directive uses `0.0.0.0:80`
- IP with no port uses `<some_ip>:80`
- Port with no IP uses `0.0.0.0:<some_port>`

IP takes priority over port

`listen` Example

If example.com is hosted on port 80 of 192.168.1.10, the first block serves the response

```
server {
    listen 192.168.1.10;

}

server {
    listen 80;
    server_name example.com;

}
```

`server_name` Directive

Used to differentiate between matching IP:Port specificity

- Checks the "Host" header field
- Matching Rules (in order):
  - Exact Match
  - Leading Wildcard
  - Trailing Wildcard
  - Regex

`server_name` Example 1

If "Host" value matches "host1.example.com" exactly, second block serves response

```
server {
    listen 80;
    server_name *.example.com;

}

server {
    listen 80;
    server_name host1.example.com;

}
```

`server_name` Example 2

If "Host" value is "www.example.org", second block serves response — if no match, then trailing wildcards take over

```
server {
    listen 80;
    server_name www.example.*;

}

server {
    listen 80;
    server_name *.example.org;

}

server {
    listen 80;
    server_name *.org;
}
```

`server_name` Example 3

The FIRST matching regex has priority. If the request is "www.example.com", the first block serves the response

```
server {
    listen 80;
    server_name ~^(www|host1).*\.example\.com$;

}

server {
    listen 80;
    server_name ~^(subdomain|set|www|host1).*\.example\.com$;

}
```

IP vs. `server_name` Recap

If NGINX cannot determine which server to select, it will choose the first block in the configuration

```
server {
    listen  192.168.1.1:80;
    server_name www.example.*;
}

server {
    listen  192.168.1.1:80;
    server_name *.example.net;
}

server {
    listen  192.168.1.1:80;
    server_name ~^(www|host1).*\.example\.com$;
}
```

default_server

- Checks against "HOST" header field
- Overrides "first-found" algorithm
- Only one declaration per IP:Port combo

```
server {
    listen  80 default_server;
    server_name example.net www.example.net;
}
```

Bad Requests

`server_name` with an empty string value can prevent bad requests. 444 response code closes TCP connection

```
server {
    listen  80;
    server_name "";
    return 444;
}
```

Location Block Example

Breaks down request further by URI

```
location /application1 {

}
```

Two most common types:

- **Prefix**
- **Regex**

Prefix Location

- Checked first, then longest match serves response
- Nested inside `server` context

```
location /application1 {

}

location /application1/images {
        alias /media/data;
}
```

Second prefix serves response if request is:

```
$ curl http://somedomain.com/application1/images/?img2
```

Location Modifiers

- Exact String Match `=`
- Case Insensitive Regex `~*`
- Case Sensitive Regex `~`
- Stop Request Processing `^~`
- Named Location Routing `@`

Regex Location

Matched sequentially and only after prefix locations.

```
location /application1 {

}

location ~* ^\.(gif|jpg|jpeg|png)$ {
        alias /media/data;
}
```

Location Order

Configuration Example

```
server {
    listen 80 default_server;
    root /usr/share/nginx/html;

    location = / {
    }

    location ~* ^\.(png|jpg)$ {
    }

    location ^~ /app1 {
    }
}
```

Selection Order:

- Location 1
- Location 3
- Location 2

Lab 1.1: Serve Pages and Images

Note: commands in "/etc/nginx/conf.d" require "sudo" or "root" level privileges

1. Navigate to the NGINX configuration directory:

```
$ cd /etc/nginx/conf.d
```

2. Backup the default and ssl configurations:

```
$ sudo mv default.conf default.conf.bak
$ sudo mv example_ssl.conf example_ssl.conf.bak
```

3. Create a new configuration called `server1.conf`:

```
$ sudo vim server1.conf
```

Lab 1.2: Serve Pages and Images

1. Create the following `server` block:

```
server {
    listen 80;
    root /home/student1/public_html;
}
```

2. Add the following three `location` prefixes:

```
location /application1 {}
location /application2 {}
location /images {}
```

3. Add an overriding `root` directive in `/images`:

```
root /data;
```

PROXYING CONNECTIONS

Module Objectives

This module enables you to:

- Configure Proxy Server
- Understand how Proxy Buffering works
- Demonstrate use of `proxy_pass` directive

Reverse Proxy Servers

Receives requests, passes them to backend servers

NGINX supports proxy for HTTP, HTTPS, TCP, UDP, and other protocols.

```
server {
    listen  80;
    server_name mydomain.com;

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

`proxy_pass` Directive

Sets the address, and protocol, of the proxied server(s)

Syntax:

```
proxy_pass $protocol$host$uri
```

Example:

```
location / {
        proxy_pass http://backend:8080/application/;
}
```

Proxy With a Path

Replaces matched portion of the request uri, including location prefix:

```
#curl request example:
$ curl http://server/application1/

#NGINX selects a matching location prefix
location /application1 {
    proxy_pass http://localhost:8080/otherapp;
}

#proxy request becomes
http://server:8080/otherapp/app1.html;
```

Proxy Without a Path

Replaces matched portion of the request uri, NOT including location prefix:

```
#curl request example:
$ curl http://server/application2/

#NGINX selects a matching location prefix
location /application2 {
    proxy_pass http://localhost:8080
}

#proxy request becomes
http://server:8080/application2/index.html
```

To / or Not To /

Beware of trailing slash request mismatches.

```
#Example curl request with NO trailing slash
$ curl http://example.com/app1
#NGINX Proxy prefix with a trailing slash, but proxy_pass has none
location /app1/ {
    proxy_pass http://application1;
}
#NGINX Backend prefix with a trailing slash
location /application1/ {
}

#NGINX log shows it appending the slash through too many redirects
GET /app1 HTTP/1.1 301 /app1/
GET /app1// HTTP/1.1 301 /app1//
...
#Correct response(s) should be
GET /app1/ HTTP/1.1 200 /app1/
GET /app1/ HTTP/1.1 304 /app1/
```

Lab 2.1: Setup a Reverse Proxy

1. Create a new configuration file called `server2.conf`:

```
$ sudo vim /etc/nginx/conf.d/server2.conf
```

2. Define the following server context:

```
server {
    listen 90;
    root /data/server2;
}
```

3. Save and exit the file, then open `server1.conf`:

```
$ sudo vim /etc/nginx/conf.d/server1.conf
```

Lab 2.2: Test Your Proxy

1. Add a `proxy_pass` in `/application1` with the URI:

```
location /application1 {
    proxy_pass http://localhost:90/sampleApp/;
}
```

2. Save and Reload NGINX

```
$ sudo nginx -s reload
```

3. Open your browser (or use `curl`) and test `http://ec2-hostname/application1`, what do you see?

```
$ curl http://ec2-hostname/application1
```

Proxy Buffers

- `proxy_buffering` *`on`* | *`off`* sets proxy buffering
- `proxy_buffers` *`number size`* sets amount and size used for reading entire response for one connection
- `proxy_buffer_size` sets size for reading first part of a response (usually response headers) received from proxied server

`proxy_busy_buffers`

This directive sets max size of “client-ready” buffers.

The “client-ready” buffers are then placed in a queue.

Proxy Busy Buffers Example

This example increases available buffers per request, while trimming down buffers stored in responses

```
server {
    proxy_buffering on;
    proxy_buffer_size 1k;
    proxy_buffers 24 4k;
    proxy_busy_buffers_size 8k;
    proxy_max_temp_file_size 2048m;
    proxy_temp_file_write_size 32k;

        location / {
            proxy_pass http://example.com;
        }
}
```

LOGGING

Module Objectives

This module enables you to:

- Setup Logging to audit connections to NGINX
- Demonstrate use cases for log levels
- Differentiate between access and error log

`error_log` Directive

- Configures the logging settings for error messages
- Syntax: `error_log` *`file log_level`*
- "Log Level" specifies the detail of the output

Log Levels

`debug`	Detailed Trace
`info`	General Info
`notice`	Something Normal
`warn`	Something Strange
`error`	Unsuccessful
`crit`	Important Issue(s)
`alert`	Fix Now!
`emerg`	Unusable

`access_log` Directive

- Records all attempts to access the server
- Default log type is *`combined`*

```
#Example
access_log /var/log/nginx/server3.access.log combined;
```

`log_format` Directive

- Defined in the `http` context
- Uses pre-defined or custom variables
- Overrides default log type i.e. `combined`

```
log_format test_log "$request $status $request_uri";

server {
    listen 80;
    root /home/student1/public_html;
    access_log /var/log/nginx/public.log test_log;
}
```

`access_log` Locations

- Default locations
  - From source: `logs/access.log`
  - Binary distro: `/var/log/nginx/access.log`

Reading the Logs

When log values are separated by:

- `,` = request processed by multiple servers
- `;` = internal redirect between upstreams
- `0` = unable to reach upstream
- `-` = internal error or cached value

Logging Best Practices

Keep a separate log files for each server

```
server {
    server_name server1.com;
    root /data/server1.com;
    error_log logs/server1.error.log    info;
}

server {
    server_name server2.com;
    root /data/server2.com;
    error_log logs/server2.error.log    info;
}
```

Rotating Logs

Run this shell script in a *cron* job

```
#Get Yesterday's date as YYYY-MM-DD
YESTERDAY=$(date -d 'yesterday' '+%Y-%m-%d')
PID_FILE=/run/nginx.pid
LOG_FILE=/var/log/error.log
OLD_FILE=/var/log/error-$YESTERDAY.log

#Rotate yesterday's log.
mv $LOG_FILE $OLD_FILE

#Tell nginx to open the log file
kill -USR1 $(cat $PID_FILE)
```

Documentation: Log Rotation

`syslog` Protocol

Use this protocol when sending messages to syslog servers such as: *splunk*, or *syslog-ng*

```
error_log syslog:server=192.168.1.1 debug;

access_log syslog:server=unix:/var/log/nginx.sock,nohostname;
access_log syslog:server=[2001:db8::1]:12345,facility=local7,tag=nginx,se
```

Documentation: Syslog

Lab 3.1: Setup Logging

1. Open `server1.conf`

```
$ sudo vim /etc/nginx/conf.d/server1.conf
```

2. Add an `error_log` and `access_log` directive with log levels of `info` and `combined`:

```
error_log /var/log/nginx/server1.error.log info;
access_log /var/log/nginx/server1.access.log combined;
```

3. Repeat the same steps for `server2.conf`
4. Save and Reload NGINX

```
$ sudo nginx -s reload
```

Lab 3.2: Test Logging

1. Open a browser and test your instance:

```
$ curl http://ec2-hostname/
$ curl http://ec2-hostname:90/
```

2. Check the logs with the `tail -f` command:

```
$ sudo tail -f /var/log/nginx/server1.access.log
$ sudo tail -f /var/log/nginx/server2.access.log
```

SECURITY

Module Objectives

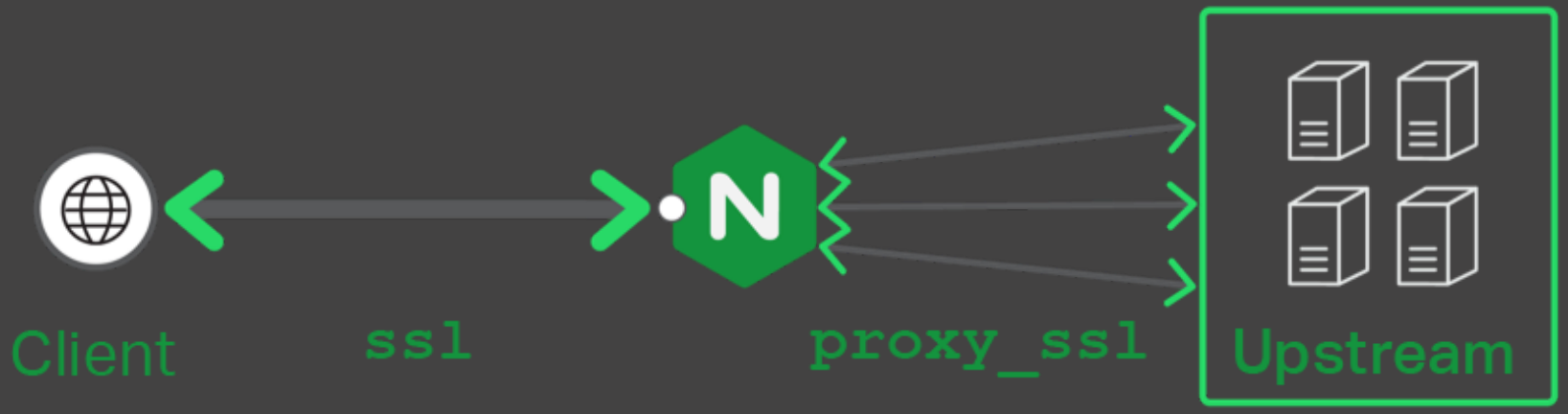
This module enables you to:

- Signature Strength
- Cipher Strength
- A+ Score on SSL Labs.com

`http_ssl_module`

- Provides directives for configuring and managing HTTPS servers
- Requires `openssl` Library

`ssl` vs. `proxy_ssl`



SSL Termination

SSL terminates at NGINX level, NGINX handles handshake overhead to take load off the backends

```
http {
    ssl_session_cache   shared:SSL:10m;
    ssl_session_timeout 10m;

    server {
        listen              443 ssl;
        server_name         www.example.com;
        keepalive_timeout   70;

        ssl_certificate     www.example.com.crt;
        ssl_certificate_key www.example.com.key;
        ssl_protocols       TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers         HIGH:!aNULL:!MD5;
        ...
    }
}
```

Documentation: SSL Termination

Popular Use Cases

Combine HTTP & HTTPS

```
server {
    listen 443;
    listen 80;
    server_name example.com;
    root /home/student1/public_html;
}
```

Popular Use Cases

Force incoming traffic to HTTPS

```
#server 1
server {
    listen 80;
    return 301 https://$host$request_uri;
}

#server 2
server {
    listen 443 ssl;
    ssl_certificate /etc/nginx/ssl/server.crt;
    ssl_certificate_key /etc/nginx/ssl/server.key;
}
```

General Infrastructure Security

- Turn off server_tokens
- Turn off corresponding backend engine headers
  - X-Powered-By
- Change client side error pages
- Encrypt ALL THE THINGS!!!
- Test on SSL Labs.com

Getting a Perfect SSL Score

- Verify cert/chain in order and from trusted authority
- Use strong signature algorithm
- Use the latest protocol support
- Generate strong key signature
- Use preferred ciphers

Configure HTTPS

- Enable `ssl` on `listen` directive
- Specify `ssl_certifcate` and `ssl_certificate_key`

```
server  {
    listen  443 ssl;
    root /data;

    ssl_certificate /etc/nginx/ssl/nginx.crt;
    ssl_certificate_key /etc/nginx/ssl/nginx.key;

}
```

ssl_dhparam

- Specifies a file with DH parameters for DHE ciphers
- Aids in Forward Secrecy
- Make them strong, 4096 rather than 2048

```
server  {
    listen  443 ssl;
    root /data;

    ssl_certificate /etc/nginx/ssl/nginx.crt;
    ssl_certificate_key /etc/nginx/ssl/nginx.key;
    ssl_dhparam ssl/dhparam.pem;
}
```

ssl_protocols

- Specifies which protocols are enabled
- Best Practice: Only support TLS
- Best Practice: Only support latest for higher rating

```
server  {
    listen  443 ssl;
    root /data;

    ssl_certificate /etc/nginx/ssl/nginx.crt;
    ssl_certificate_key /etc/nginx/ssl/nginx.key;
    ssl_dhparam ssl/dhparam.pem;
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
}
```

ssl_ciphers

- Specifies a list based on business needs
- Moving target; ciphers constantly change, new security threats arise daily
- Configure NGINX to force client to accept preferred order of ciphers

```
server  {
    listen  443 ssl;
    root /data;

    ssl_certificate /etc/nginx/ssl/nginx.crt;
    ssl_certificate_key /etc/nginx/ssl/nginx.key;
    ssl_dhparam ssl/dhparam.pem;
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers AES256+EECDH:AES256+EDH:!aNULL;
    ssl_prefer_server_ciphers on;
}
```

ssl_stapling

- `ssl_stapling` ”staples” an OCSP response
- `ssl_stapling_verify` verifies OCSP responses
- `ssl_trusted_certificate` required

```
server  {
    listen  443 ssl;
    root /data;

    ssl_certificate /etc/nginx/ssl/nginx.crt;
    ssl_certificate_key /etc/nginx/ssl/nginx.key;
    ssl_dhparam ssl/dhparam.pem;
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers AES256+EECDH:AES256+EDH:!aNULL;
    ssl_prefer_server_ciphers on;
    ssl_stapling on;
    ssl_stapling verify on;
}
```

ssl_session

- ssl_session_cache
- ssl_session_timeout
- ssl_handshake_timeout

HSTS

- Forces browsers to communicate over `https`
- Other Security Precautions:
  - No-Content-Sniffing
  - No content displaying in iFrames

```
add_header Strict-Transport-Security max-age=63072000;
add_header X-Content-Type-Options nosniff;
add_header X-Frame-Options DENY;
```

"Dual Stack" RSA and ECC

- ECC is 3x faster than RSA
- Include both pairs to support both

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

    ssl_certificate example.com.rsa.crt;    ssl_certificate_key example.
    ssl_certificate example.com.ecdsa.crt;  ssl_certificate_key example.
}
```

Elliptic Curves

NGINX doesn’t have a good default for elliptic curves, so specify manually e.g.

```
ssl_ecdh_curve secp384r1;
```

Lab 4.1: SSL Certs

1. Create a directory for certs and keys

```
$ sudo mkdir /etc/nginx/ssl -p
```

2. Change directories and run the openssl command:

```
$ cd /etc/nginx/ssl
$ sudo openssl req -x509 -nodes -days 365 \
-newkey rsa:2048 -keyout nginx.key -out nginx.crt \
```

Lab 4.2: Configure SSL Parameters

1. Open `server1.conf`, and add the following directives in the `server` context:

```
ssl_certificate /etc/nginx/ssl/nginx.crt;
ssl_certificate_key /etc/nginx/ssl/nginx.key;

ssl_protocols TLSv1.2;
ssl_ciphers "AES256+EECDH:AES256+EDH:!aNULL";
ssl_prefer_server_ciphers on;
ssl_ecdh_curve secp384r1;
ssl_session_cache shared:SSL:10m;
ssl_session_timeout 10m;
ssl_session_tickets off;

add_header Strict-Transport-Security
    "max-age=63072000; includeSubdomains";
add_header X-Frame-Options DENY;
add_header X-Content-Type-Options nosniff;
```

2. Save and Close the file

Lab 4.3: Configure HTTPS

1. Convert your `server` context to listen on `433` with `ssl`:

```
server {
    listen 443 ssl;
    root /home/student1/public_html;
    error_log /var/log/nginx/server1.error.log info;
    access_log /var/log/nginx/server1.access.log combined;
    ...
    }
}
```

2. Add a new `server` block forcing `http` traffic to `https`:

```
server {
    listen 80
    return 301 https://$host$request_uri;
}
```

Lab 4.4: Test on SSL Labs.com

1. Save and `reload` NGINX
2. Test your site on SSL Labs.com
3. Share your results with the class

Lab 4.5: What the "F"

1. Your test should recieve an "F" rating
2. To patch the Oracle padding error, run the following commands:

```
$ sudo apt-get install --only-upgrade libssl1.0.0 openssl
$ sudo nginx -s stop
$ sudo apt-get update
$ sudo nginx
```

3. Clear your cache in SSLlabs.com, and re-test
4. You should receive a "T" rating

Restricting Access

- `allow` specifies access to a server or location block
- `deny` directive prevents access

Basic HTTP Authentication

Requires users to specify a login and password via the browser. Syntax is:

```
auth_basic realm | off
```

Basic Auth Example

```
server {
    listen 8080;
    root /home/johnny/public_html;

    auth_basic “restricted server”;
    auth_basic_user_file /etc/nginx/htpasswd;

    location /public_area {
        auth_basic off;
    }
}
```

Password File

- `crypt()`
- `openssl passwd`

```
name1:password1
name2:password2
#comments
```

`auth_request` Module

Implements authentication based on the result of a sub-request, rather than the browser

```
server {
    listen 443 ssl;
    root /usr/share/nginx/html;
...
    location /private {
        auth_request /auth;
        auth_request_set  $user $upstream_http_x_user;
        proxy_set_header x-user $user;
        proxy_pass http://ldap_backend;
    }
}
```

Lab 5.1: Create a User:Password

1. Open the password file called `htpasswd` inside `/etc/nginx`

```
$ sudo vim /etc/nginx/htpasswd
```

2. Delete the password for the admin entry
3. Close and save the file. Back in the terminal, run the following command:

```
$ openssl passwd
```

4. Specify any user and password, copy the encrypted output to your clipboard, and paste it back into `htpasswd`

Lab 5.2: Setup Basic Auth

1. Open `server1.conf` and enable authentication on the `server` context. Then point to the `htpasswd` file:

```
$ sudo vim /etc/nginx/conf.d/server1.conf

server {
    listen 443 ssl;
    ...
    auth_basic "protected";
    auth_basic_user_file /etc/nginx/htpasswd;
}
```

2. Save and reload NGINX. Open a browser and test your instance's home page
3. Login with your username and password you specified

Setup Limit Rates

Limitations based on:

- Number of Connections
- Number of Requests
- Download Speeds

Limiting Connections

- Define a Shared Memory Zone
- `limit_conn_zone` defines: zone, size, name, and key
- Syntax is:

```
limit_conn_zone $variable zone=name:size;
```

limit_conn

Specify shared memory zone in desired context

Syntax:

`limit_conn zone number_of_connections;`

```
http {
    limit_conn_zone $binary_remote_addr zone=ip:10m;

    server {
        …
        location /downloads {
                limit_conn ip 5;
    }
```

Limiting Request Rate

Set request rate, name, key, and duration in Shared Memory Zone

Example:

```
limit_req_zone $server_name zone=one:10m rate=1r/s
```

limit_req

Syntax:

```
limit_req zone='name' [burst=n.o | nodelay];
```

```
http {
        limit_req_zone $server_name zone=ten:10m rate=10r/s
        …
        server {
                …

                location /data {
                        limit_req zone=ten burst=15;
                }
        }
}
```

Limiting Download Rate

Syntax:

`limit_rate` *`speed`*

```
location /media {
    limit_conn ipzone 5;
    limit_rate 50k;
}
```

limit_rate_after

Throttle download speeds after client builds a buffer

```
location /videos {
    limit_rate_after 500k;
    limit_rate 50k;
}
```

VARIABLES

Module Objectives

This module enables you to:

- Gain knowledge of NGINX's predefined variables
- Understand Variable Scope with regards to NGINX
- Define your own custom variables

Core Module Variables

Variable	Value
`$host`	Host name from request line
`$request_uri`	The full URI, including arguments
`$uri`	Normalized URI (no arguments)
`$scheme`	Request scheme (HTTP or HTTPS)
`$request_method`	GET, POST, PUT etc.
`$request_filename`	Absolute file path for current request
`$remote_addr`	IP address of client

Variable Example

Given URL: http://localhost:8080/test?arg=1

- `$host` = localhost:8080
- `$request_uri` = /test?arg=1
- `$uri` = /test
- `$scheme` = http://
- `$args` = ?arg=1

Usage Example:

```
server {
    listen 80;
    return 301 https://$host$request_uri;
}
```

Useful Troubleshooting Variables

- `$upstream_connect_time`
- `$upstream_header_time`
- `$upstream_response_time`
- `$request_time`

`set` Directive

Syntax:

```
set $variable_name value;
```

```
set $foo hello;
set $bar “hello world”;
set $combo $foo
```

Declaration and Scope

- Declared when NGINX loads the configuration file
- Assigned at runtime
- Every config can have access to a variable
- Similar to Java method invocation

Scope Example

```
server {
    listen 8080;

        location /test1 {
            return 200 = "foo $foo \name example = $example \n";
        }

        location /test2 {
            set $foo hello;
            return 200 = "foo = $foo \n";
        }
}

server {
    listen 8081;

    set $example 42;
```

`map` Directive

Creates a mapping relationship between two variables

Syntax:

```
map $var1 $var2 {value value}
```

```
map $uri      $path {
    /test1    /path1
    /test2    /path2
    /test3    /path3
}

server {
        listen 80;

        location /test1 {
        return 200 "$path \n";
    }
}
```

Default Value

Default values protect NGINX from translation errors

```
map $arg $value {
    default 1;
    test1 2;
}
```

Regex in Maps

Only named captures work in maps, no positional captures

```
map $uri $path {
    ~*/test/.*\.php$ /path4;
    /example /examplePath;
}
```

Use Case: Conditional Logging

Use variables to determine "loggability"

```
map $status $loggable {
    ~^[23]  0;
    default 1;
}

access_log /path/to/access.log combined if=$loggable;
```

Lab 6: Map Example

1. Create a `map` for variables `$is_redirect` and `$uri`:

```
map $uri $is_redirect {
    default   0;
    /test1    1;
    /test2    2;
    /test3    3;
}
```

2. Create a regex for /test:

```
server {
        ....
        location ~* /test(\d+)$ {
        return 200 "variable = $is_redirect \n";
    }
}
```

3. Test the map using curl. For example:

```
curl http://localhost/test1
```

END OF PART ONE:

THANK YOU!

Agenda: Part Two

- Routing Connections
- Load Balancing
- Caching
- Compression
- Dynamic Configuration
- Installation

ROUTING CONNECTIONS

Module Objectives

This module enables you to:

- Use specific directives in NGINX to reroute traffic
- Define URL Rewrites
- Understand Rewrite Request processing

`alias` Directive

Allows you to specify a replacement path

Syntax:

```
alias path;
```

```
server {
    root /home/public_html;

    location /test {
            alias /data/app1;
        }
}
```

Lab 7: Alias Replacement

1. Open `server1.conf`:

```
$ sudo vim /etc/nginx/conf.d/server1.conf
```

2. In the `/application2` prefix, specify a replacement path of `/data/test` using the `alias` directive:

```
location /application2 {
    alias /data/test;
}
```

3. Save the file and reload Nginx. Test against `http://server/application2/logo.png` What do you observe? Try the other URIs

`alias` with Regex

```
location ~ ^/pictures/(.+\.(?:gif|jpe?g|png))$ {
    alias /data/images/$1;
    }
```

Lab 7.5: Regex Alias

1. Open `server1.conf` and edit the `/images` prefix to be a case insensitive regex with a URI match of `/pictures`:

```
location ~ ^/pictures/(.+\.(gif|jpe?g|png))$ {
    alias /data/images/$1;
}
```

2. Save the file and reload Nginx. Test against `http://server/pictures/logo.png` What do you observe?

`return` Directive

Return a HTTP response code and URL to the client

Syntax:

`return` *`code url;`* or `return` *`url;`*

```
server {
    listen 8080;
    root /home/public_html;

    location /test {
            return http://localhost:8081$uri;
        }

server {
    listen 8081;
    root /data/app1;
    autoindex on;
    }
}
```

`rewrite` Directive

Regex pattern matched against URI, replacement string rewrites the URI.

Syntax:

```
rewrite regex replacement [flag]
```

```
server {
    listen 8080;
    root /home/public_html;
    rewrite ^/shop/products/(\d+) /myshop/products/product$1.html;
}
```

Lab 8.1: Setting Up Rewrite Data

1. Change directories to:

```
$ cd /home/student1/public_html/shop/products
```

2. Ensure that the following files exist:

```
$ ls
product1.html product2.html product3.html
```

3. Open `product1.html` amd edit the paragraph tag by removing `8080` and replacing `<server>` with your ec2-hostname

```
<p>
<img src="http://<ec2-hostname>/media/pics/logo.png"/>
</p>
```

Lab 8.2: Rewrite URLs

1. Open `server1.conf` in `/etc/nginx/conf.d`
2. Inside the `server` context, define a `rewrite` regex:

```
^/shop/greatproducts/(\d+)$
```

   and a replacement string:

```
/shop/products/product$1.html
```

3. Save your file and reload nginx
4. Open your browser and test against:
   - `<server>/shop/greatproducts/2`
   - `<server>/shop/greatproducts/3`
   - `<server>/shop/greatproducts/1`

Lab 8.3: Rewrite URLs (Continued)

1. Now try `<server>/shop/greatpoducts/1`
2. What do you notice about the image?
3. Add another `rewrite` at the `server` level with the following regex:

   ```
   ^/media/pics/(.+\.(gif|jpe?g|png))$
   ```

   and replacement string:

   ```
   /images/$1
   ```

4. Save and reload NGINX
5. Re-test `/shop/greatproducts/1`
6. If the image still doesn't render, ensure you have a prefix or regex location enabled for your image content

Rewrite Process Cycle

Highlights:

- Executed sequentially
- Executed upon server selection
- All rewrites in higher context are executed first
- Flags will stop further processing

`rewrite flags`

Prevents further rewrite processing

- `last`
- `break`
- `redirect`
- `permanent`

Lab 9.1: Rewrite Flags

1. Open `server1.conf` and define a new location block with the prefix `/shop`
2. Cut and paste the `rewrite` regarding `/greatproducts`, and paste it into the new `location`
3. Add a new `rewrite` with regex:

   ```
   ^/shop/.+/(\d+)$
   ```

   and replacement string:

   ```
   /shop/services/service$1.html
   ```

4. Add a `return` directive with code `403`
5. Save and reload NGINX. Re-test `/shop/greatproducts/1`

Lab 9.2: Rewrite Flags (Continued)

1. Re-open `server1.conf` and place `break` flags after every `rewrite` in the `location` context
2. Save and reload NGINX
3. Re-test `/shop/greatproducts/1`

LOAD BALANCING

Module Objectives

This module enables you to:

- Setup basic load balancer
- Identify load balancing methods
- Enable session persistence

`ngx_http_upstream_module`

Key Take-Aways:

- Defines a group of servers
- Leverages `proxy_pass` directive
- Server definition can be:
  - Unix socket
  - DNS
  - IP:Port

Load Balancing

`proxy_pass` forwards request to `upstream` link

```
upstream myServers {
       server training.example.com;
       server training.example1.com:8080;
       server 192.168.245.27;
          }
```

Specifying Server Priorities

`weight` indicates `server` priority

`max_fails` indicates server-level failures

`fail_timeout` indicates timeout and duration of downtime

```
upstream myServers {
       server backend.server1 weight=5 max_fails=10 fail_timeout=90s;
       server backend.server2 weight=3 max_fails=4 fail_timeout=60s;
       server backend.server3 weight=4 max_fails=2 fail_timeout=30s;
          }
```

Lab 10.1: Setup "Backends"

1. In the `/etc/nginx/conf.d` , create a new .conf called `backends.conf` and define a `server` context with the `root` directive pointing to `/data/backend1`. The server should listen on `8081`
2. Repeat steps 1 and 2 for the remaning servers. Servers should listen on `8082` and `8083` respectively, and the `root` directories are `/data/backend2` and `/data/backend3`

Lab 10.2: Configure Load Balancing

1. Create a file called `myServers.conf` and define an `upstream` with three backend servers using `127.0.0.1:<port>`
2. Create a `server` context that listens on `8080`
3. Set the `root` directive to your `public_html` directory.
4. Define an `error_log` with a level of `info` and an `access_log` with a level of `combined`
5. Create a `location` context that matches all requests
6. Add a `proxy_pass` to forward all request to the `upstream`

`ngx_stream_core_module`

Key Take-Aways:

- Used for TCP/UDP Load Balancing
- Also leverages `proxy_pass` directive
- Similar syntax with `ngx_http_upstream` module
- Version compatibility:
  - TCP: r5 or greater
  - UDP: r9 or greater
- Exists in `main context`

`stream` Use Cases

- **TCP:** mSQL, LDAP, RTMP
- **UDP:** DNS, Syslog, RADIUS

mySQL Example

Load Balancings across three SQL servers

```
stream {
    server {
            listen 3306;
            proxy_pass db;
    }

    upstream db {
            server db1:3306;
            server db2:3306;
            server db3:3306;
    }
}
```

DNS Example

Load balances UDP traffic across two DNS servers

```
stream {
    upstream dns_upstreams {
            server 192.168.136.130:53;
            server 192.168.136.131:53;
    }

    server {
            listen 53 udp;
            proxy_pass dns_upstreams;
            proxy_timeout 1s;
            proxy_responses 1;
            error_log logs/dns.log;
    }
}
```

Load Balancing Methods

- `least_conn`
- `least_time`
- `hash`
- `ip_hash`

`least_conn` Directive

```
upstream backendServers {
        least_conn;

        server backend1.com;
        server backend2.com;
        server backend3.com;
}
```

`least_time` Directive

```
upstream myServers {
        least_time header;

        server backend1.com;
        server backend2.com;
        server backend3.com;
}
```

`hash` Directive

```
upstream myServers {
        hash $request_uri;

        server backend1.com;
        server backend2.com;
        server backend3.com;
}
```

`ip_hash` Directive

Uses first three octets for IPv4, or entire IPv6 address

```
upstream myapp1 {
    ip_hash;
    server srv1.example.com;
    server srv2.example.com;
    server srv3.example.com;
}
```

`fail_timeout` Parameter

```
upstream myServers{
    server backend1.example.com:8080 max_fails=3 fail_timeout=30s;
        …
}
```

`max_conns` Parameter

```
upstream backend {
        server backend1.example.com max_conns=3;
        server backend2.example.com;
}
```

`queue` Directive

```
upstream backend {
        server backend1.example.com max_conns=3;
        server backend2.example.com;
                queue 100 timeout=70;
}
```

Session Affinity

For applications that require state data on backend servers

NGINX supports the following methods:

- `sticky cookie`
- `sticky learn`
- `sticky route`

HTTP using DNS

NGINX Plus can monitor changes of IP addresses that correspond to domains and resolve them

```
http {
    resolver 10.0.0.1 valid=300s ipv6=off;
    resolver_timeout 10s;

    server {
        location / {
            proxy_pass http://backend;
        }
    }

    upstream backend {
        zone backend 32k;
        least_conn;
        ...
        server backend1.example.com resolve;
        server backend2.example.com resolve;
    }
}
```

Load Balancing NTLM

Release 7+ only. Requires HTTP 1.1 connection

```
http {
    ...
    upstream exchange {
        zone exchange 64k;
        ntlm;
        server exchange1.example.com;
        server exchange2.example.com;
    }

    server {
        listen              443 ssl;
        ssl_certificate     /etc/nginx/ssl/company.com.crt;
        ssl_certificate_key /etc/nginx/ssl/company.com.key;
        ssl_protocols       TLSv1 TLSv1.1 TLSv1.2;

        location / {
            proxy_pass         https://exchange;
            proxy_http_version 1.1;
```

Lab 11: Configure Hash Method

1. Open `myServers.conf` and define a `hash` selection algorithm in `upstream myServers`
2. Save and reload nginx, test in browser several times
3. Notice how the session sticks to that particular server?
4. Now run a dynamic curl request from a local terminal

```
for i in `seq 1 100` ; do curl -s -o /dev/null -w "%{http_code}" \
http://<external ip>/\?$i \
; done
\
```

5. Check the log files; are the connections still persistent?

LIVE ACTIVITY MONITORING

Module Objectives

This module enables you to:

- Use the `status` directive to get server metrics
- Configure `health_check` to monitor the availability of your upstream servers

`status` Directive

```
server {
    listen 8080;

    location = /status{
            status;
    }
}
```

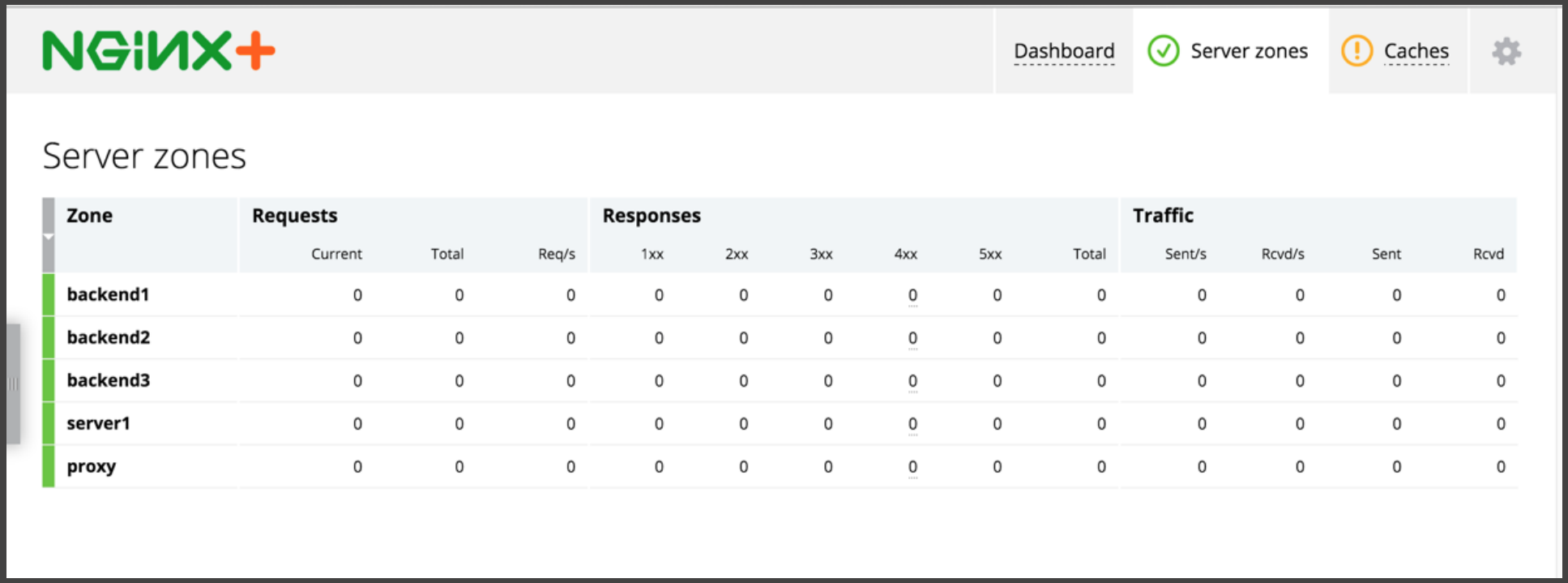
Securing the Status Request

```
location = /status{
    allow 192.168.0.0/16;
    deny all; # deny access from everywhere else

    status;
}
```

Default Status Page

NGINX Plus contains an html page that parses JSON

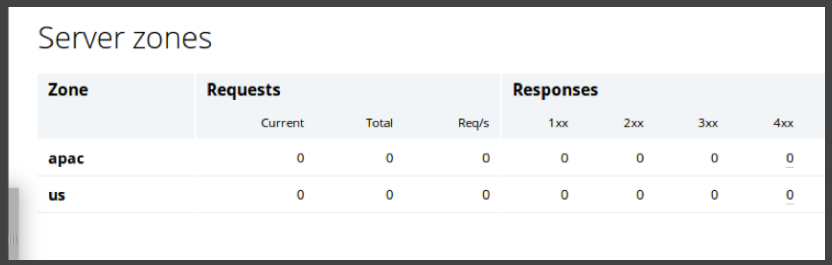
The page is located at **/usr/share/nginx/html/status.html**



Server Zones

```
server {
        listen 8081;
        root /server/backend1;
        status_zone apac;
        }

server {
        listen 8082;
        root /server/backend2;
        status_zone apac;
}

server {
        listen 8083;
        root /server/backend3;
        status_zone us;
}
```

Individual servers zones displayed in status.html:



Server zones

Zone	Requests			Responses			
	Current	Total	Req/s	1xx	2xx	3xx	4xx
apac	0	0	0	0	0	0	0
us	0	0	0	0	0	0	0

`zone` Directive

Shared memory zone makes upstream dynamically configurable

Allows worker processes to share counter information

```
upstream myServers {
    zone backend 64k;
        server backend1;
        server backend2;
        server backend3;
        }
```

Lab 12.1: Define a Status Page

1. Open `myServers.conf` located in `/etc/nginx/conf.d/`
2. In a new `server` context that listens on `9090` with a `root` location `/usr/share/nginx/html;`
3. Add a `location` prefix with an exact match `= /status`, and the `status` directive in this location block
4. Save and reload NGINX
5. Open a browser and access the following URI:

```
http://<server>:9090/status.html
```

6. Notice anything strange in the GUI?

Lab 12.2: Define Server Zones

1. In `backends.conf`, specify a server zone for each server by using the `status_zone` directive
2. Open `server1.conf` and define a `status_zone`
3. Open `server2.conf` and define a `status_zone`
4. Open `myServers.conf` and specify a shared memory zone in your `upstream` block with a size of `64k`
5. Save and reload NGINX
6. Refresh your browser listening on

```
http://<server>:9090/status.html
```

Server `health_check`

A request sent to upstream to check status based on conditions

```
upstream myUpstreams {
                zone backend 64k;

                server localhost:8081 weight=1;
                server localhost:8082 weight=4;
                server localhost:8083 weight=7;
        }

        server {
            listen 8080;
            error_log /var/log/nginx/upstream_error.log info;

            location / {
                proxy_pass http://myUpstreams;
                health_check;
            }
}
```

`health_check` Parameters

- `interval`
- `fails`
- `passes`
- `uri`
- `match`

`match` Block

Block directive that defines conditions for:

`health_check`

Conditions can be based on:

- Response Codes
- Header Values
- Text body of documents

`match` Directives

- `status`
  - `status 200`
  - `status ! 403`
  - `status 200-399`
- `header`
  - `header Content-Type = text/html`
  - `header Cache-Control`
- `body`
  - `body ~ "Hellow World"`
  - `body !~ "hello world"`

match Example

```
server {
    location / {
    proxy_pass http://myServers;
    health_check match=conditions fails=2;
    }
}
    match conditions {
    status 200;
    header Content-Type = text/html;
    body !~ “maintenance”;
}
```

Lab 13.1: Server Maintenance

1. Open the `myServers.conf`, file
2. Change your upstream `server` entries back to `localhost:8081, localhost:8082, localhost:8083`
3. Define a `match` block called `health_conditions` with the following directives:

```
match health_conditions {
    status 200-399;
    header Content-Type = text/html;
    body !~ maintenance;
}
```

Lab 13.2: Health Check Params

1. In the `/`, add the following `match` parameters:

```
location / {
    proxy_pass http://myServers;
    health_check match=health_conditions
    fails=2
    uri=/health/test.html;
}
```

2. Save and reload NGINX
3. Refresh your browser listening on

```
http://<server>:9090/status.html
```

4. A server is down. To fix it, change the text "maintenance" in the test.html for backend3

```
$ cd /data/server2/backend3/health/
$ sudo vim test.html
```

CACHING

Module Objectives

This module enables you to:

- Define a reverse proxy cache for your upstream and other servers
- Purge old or stale content from the cache
- Identify other cache control techniques

Reverse Proxy and Caching

Common use case to have NGINX in front, caching static resources to improve performance

To compose a cache:

- Define a `cache_path`
- Configure the `proxy_pass`
- Reference the `cache_key`
- Validate the cacheability of content using `proxy_cache_valid`

proxy_cache_path

- Defined in `http` context
- Directive must point to directory or mount point/volume attached to instance
- Uses hybrid persistent model via a two level directory system

```
proxy_cache_path /data/nginx/cache levels=1:2 keys_zone=img_cache:20m ina
```

Configuring the `proxy_cache`

- `proxy_cache_key`: generates md5 hash
- `proxy_cache`: writes to the cache

```
location / {
    proxy_pass http://application.com:8080;

    proxy_cache_key "$scheme$host$request_uri";
    proxy_cache my-cache;
}
```

Validating the Cache

Honors (or overrides) Cache-Control origin headers

```
location / {
    proxy_pass http://application.com:8080;

    proxy_cache_key "$scheme$host$request_uri";
    proxy_cache my-cache;

    proxy_cache_valid any 1m;
    proxy_cache_valid 404 1m;
}
```

Passing Headers

In order to audit client address, as well as proxy address, we can forward the hearders to the backend

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

`add_header` Directive

Inserts response headers—can map to a variable

```
server {
[…]
    add_header X-Proxy-Cache $upstream_cache_status;

    location / {
    proxy_cache myCache;
    proxy_pass http://localhost:8081;
        }
}
```

Lab 14.1: Proxy Cache

1. Open `server1.conf`
2. Define a cache path in the `http` context:

```
proxy_cache_path /data/nginx/cache levels=1:2
keys_zone=img_cache:20m inactive=5m;
```

3. In the `server` context, set the `proxy_cache_key` to the `$scheme`, `$host`, and `$request_uri`
4. Add the following in your `server`context

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

Lab 14.2: Proxy Cache Continued

1. Set the `proxy_cache` in the `application1` prefix, and set the validation of the cache for 10 minutes.

```
location /application1 {
        proxy_cache img_cache;
        proxy_cache_valid 10m;
        proxy_pass http://localhost:90/sampleApp
```

2. Save and reload NGINX
3. Reload NGINX and make a request to

```
http://<server>/application1
```

4. Hit refresh multiple times and then check your status.html in a separate tab. Notice the cache icon is now “warm” and the hit ratio is increasing

Lab 15.1: Proxy Upstream Cache

1. Open `Open /etc/nginx/conf.d/myServers.conf`
2. Create a `proxy_cache_path` with a `keys_zone` named `upstreamCache` that lasts for:
    - 10 minutes
    - Has a max size of 60mb
    - Timeouts after 60 minutes

Lab 15.2: Proxy Upstream Cache Continued

1. Set the `proxy_set_header` to forward the client host and IP
2. Use the `add_header` directive with the `X-Proxy-Cache` response header to return the upstream status
3. Add `proxy_cache` to the proxying location
4. Save and reload NGINX
5. Make a request to your upstream and notice the second cache in your status.html
6. Try making a `curl` request? What do you see with the `-I` parameter?

Caching Resources

Directives that control cached responses:

- `proxy_cache_min_uses`
- `proxy_cache_methods`

Caching limit rates:

- `proxy_cache_bypass`
- `proxy_no_cache`

Documentation: Cache Admin Guide

proxy_cache_min_uses

```
server {
    proxy_cache myCache;
    proxy_pass http://localhost:8081;
    proxy_cache_min_uses 5;
}
```

`cache_methods` and `no_cache`

Syntax:

`proxy_cache_methods <REQUEST METHOD>`

Syntax:

`proxy_no_cache $arg`

```
map $request_uri $no_cache;
    /default    0;
    /test               1;

server {
        proxy_cache_methods GET HEAD POST;
        proxy_no_cache $no_cache;
}
```

Cache Manager and Loader

- `loader_threshold`
- `loader_files`
- `loader_sleeps`

```
proxy_cache_path /data/nginx/cache keys_zone=one:10m loader_threshold=30(
```

`proxy_cache_purge` Directive

Allows you to remove full cache entries that match a configured value.

```
server {
    proxy_cache myCache;
    proxy_pass http://localhost:8081;
    proxy_cache_purge $purge_method;
}
```

Purge Methods

Partial Purge

- use `curl` command to send `PURGE HTTP` request, `map` evaluates request and enables the directive

Full Purge

- turn `purger` parameter on in the `proxy_cache_path`, all wildcard pages will also be purged

HTTP PURGE Example

Request:

```
$curl –X PURGE –D – “http://www.mysite.com"
```

```
# setting the default purge method will only delete matching URLs.
map $request_method $purge_method {
    PURGE 1;
    default 0;
        }
server {
    listen 80;
    server_name www.mysite.com
    proxy_cache myCache;
    proxy_pass http://localhost:8081;
    proxy_cache_purge $purge_method;
}
```

purger Example

Request:

```
$curl –X PURGE –D – “http://www.mysite.com/*"
```

```
proxy_cache_path /data/nginx/cache levels=1:2 keys=myCache:10m purger=on

server {
    listen 80;
    server_name www.mysite.com;
    location / {

        proxy_cache_purge $purge_method;
    }
}
```

Lab 16.1: Configure Cache Purge

1. Open `myServers.conf` and use a `map` to create a custom variable called `purge_method` that depends on the predefined `request_method`
2. In the `location` where caching occurs, specify a condition for the cache purge request.
3. Save and reload NGINX
4. Send the purge command using the `curl` command, your machine url, and port. E.g. `http://<server>:8080`.
5. A successful purge should return an `HTTP 204` code (no content).

COMPRESSION

HTTP `gzip` Module

Key Directives:

- gzip
- gzip_types
- gzip_proxied

`gzip` Example

```
http {
    gzip on;
    gzip_types text/plain text/css;
    gzip_proxied any;
}
```

gzip_min_length

Specifies the minimum length of the response to compress

```
gzip_min_length 1000;
```

gzip_proxied

NGINX doesn't compress proxied requests by default,

`gzip_proxied` instructs NGINX to check header fields

```
server {
gzip on;
gzip_types text/plain application/xml;
gzip_proxied no-cache no-store private expired auth;
gzip_min_length 1000;
}
```

gzip_static

Syntax:

`gzip_static on | off`

```
server {
gzip on;
gzip_static on;
gzip_types text/plain application/xml;
gzip_proxied no-cache no-store private expired auth;
gzip_min_length 1000;
}
```

HTTP `gunzip` Module

Decompresses client gzip responses, if gzip method isn't supported

Syntax:

`gunzip_buffers` *`number size`*

```
http {
gunzip on;
gunzip_buffers 32 4k;
```

gzip_vary

Places the

"Vary: Accept-Encoding" response header if both gzip_static and gunzip are active.

```
server {
        gzip on;
        gzip_vary on;
        gzip_types text/plain application/xml application/json;
        gzip_proxied no-cache no-store private expired auth;
        gzip_min_length 1000;
        }
}
```

DYNAMIC CONFIGURATION

Module Objectives

This module enables you to:

- Leverage NGINX API to dynamically configure server information
- Use the state directive to make changes persistent

Dynamic Configuration

Advantages:

- View, modify, and remove servers at runtime
- No need to reload NGINX to affect changes

Disadvantages

- Runtime config is not saved to conf file
- Changes revert back to conf settings after reload (unless using `state` directive)

Shared Memory Zone

- Required to make server dynamically configurable
- Distributes traffic more evenly
- Necessary for `state` changes (covered later)

upstream_conf Directive

```
upstream myServers {

            server localhost:8081;
            server localhost:8082;
    }

    server {
        listen 8080;

        location / {
            proxy_pass http://myServers;
    }
        location /upstream_conf {
            upstream_conf;
            allow 127.0.0.1;
            deny all;
    }
```

`upstream_conf` Parameters

Key Parameters:

- `add=`
- `remove=`
- `drain=`
- `upstream=name`
- `server=address`
- `id=number`

Sending Requests

View and modify server details

```
#View all primary servers in upstream group myServers
curl http://<server>:8080/upstream_conf?upstream=myServers

#View individual server detail in upstream group
curl http://<server>:8080/upstream_conf?upstream=myServers&id=<id number>
```

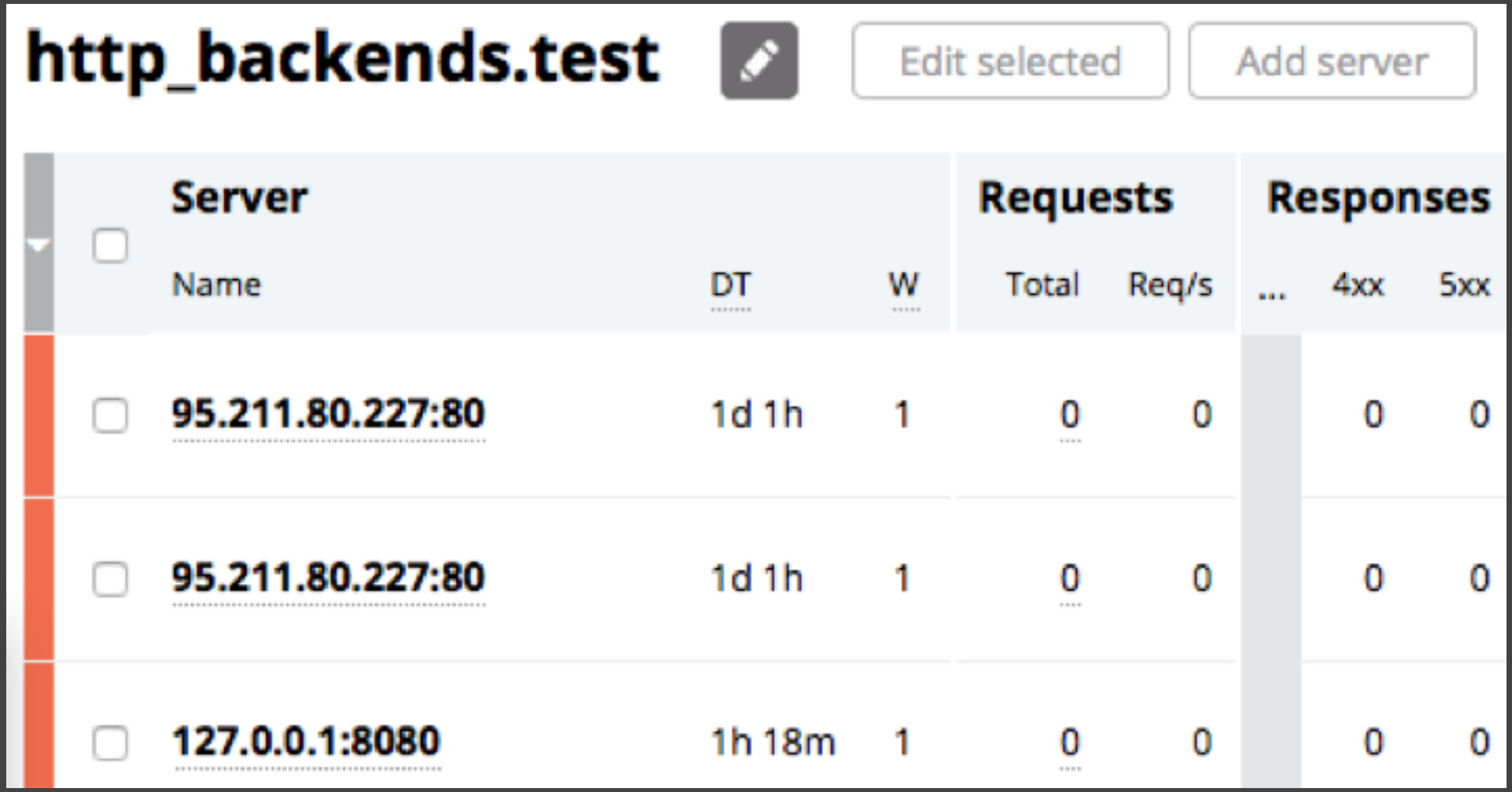
Add and Remove Servers

```
#Add a new server to the myServer group with address localhost:8083 and v
http://<server>:8080/upstream_conf?add=&upstream=myServers&server=localh

#Remove server with id=0 from the myServers upstream group
http://<server>:8080/upstream_conf?remove=&upstream=myServers&id=0
```

More Examples

```
#Modify the server with id = 0 and set the weight to 5 and the max_fails
http://<server>:8080/upstream_conf?upstream=myServers&id=0&weight=5&max_

#Modify the server with id = 0 and set the route parameter to tomcat1
http://<server>:8080/upstream_conf?upstream=myServers&id=0&route=tomcat1

#Modify the server with id = 0 and set the server address to newdomain.c
http://<server>:8080/upstream_conf?upstream=myServers&id=0&server=newdom
```

Using the Status Dashboard



Lab 17: Dynamic Config

1. Open `myServers.conf` and add an `upstream_conf` location prefix in the same server block where the `status` prefix is located.
2. Add the `upstream_conf` directive inside the `upstream_conf prefix`
3. Open your browser and hit `http://server:9090/upstream_conf?upstream=myServers`, note the id number of the first server
4. Use the API commands to `remove` the server, then `add` it back with a `weight` of 5
5. Try using the `status.html` page to change server details

Persistent Changes

For SDP protocols that allow automatic detection of devices and/or services, changes must persist across reloads

Documentation: Microservices architecture

Demo: Service Descovery with Consul

state Directive

`state` name MUST match `zone` name

Syntax:

`state` *`file/path.state`*

```
upstream myServers {
    zone backend 64k;
    state /etc/nginx/conf.d/backend.state;
}
```

Lab 18.1: Persistent Changes

1. Create a directory for the `state` file and change ownership so NGINX has write permissions:

```
$ sudo mkdir -p /var/lib/nginx/state
$ sudo chown nginx:nginx /var/lib/nginx/state
```

2. Open `server1.conf` and comment out each `server` in the `upstream`, then add the `state` directive:

```
upstream myServers {
    zone backend 64k;
    state /var/lib/nginx/state/backend.state;

    #server 127.0.0.1:8081;
    #server 127.0.0.1:8082;
    #server 127.0.0.1:8083;
}
```

Lab 18.2: Changing the State

1. Save NGINX and reload. Check `status.html`
2. Your servers vanished!
3. Create/edit the state file and add server details to create a new upstream:

```
#Create and edit the state file:
$ sudo vim /var/lib/nginx/state/backend.state

        server 127.0.0.1:8081;
        server 127.0.0.1:8082;
        server 127.0.0.1:8083;
```

4. Go to your `status.html` page and edit these server details using the GUI
5. Reload and NGINX and see if the changes persist

INSTALLATION

Module Objective

This module will enable you to:

- Install NGINX from a binary distribution
- Compile binary from source code

CentOS, RHEL

- Setup `yum` repository
- Edit repo file to pull latest packages
- Update repo
- Install NGINX

Documentation: CentOS/RHEL Install Guide

Debian, Ubuntu

- Authenticate repo signature
- Retrieve distribution components
- Resolve dependencies
- Install NGINX

NGINX Signing Key

```
wget http://nginx.org/keys/nginx_signing.key
                sudo apt-key add nginx_signing.key
```

Distribution URL

Edit the sources.list to retrieve correct distribution components

```
#Open the sources.list file with vim
sudo vim /etc/apt/sources.list

#For Debian, append the following distribution URLs
deb http://nginx.org/packages/debian/ codename nginx
deb-src http://nginx.org/packages/debian/ codename nginx

#For Ubuntu
deb http://nginx.org/packages/ubuntu/ codename nginx
deb-src http://nginx.org/packages/ubuntu/ codename nginx
```

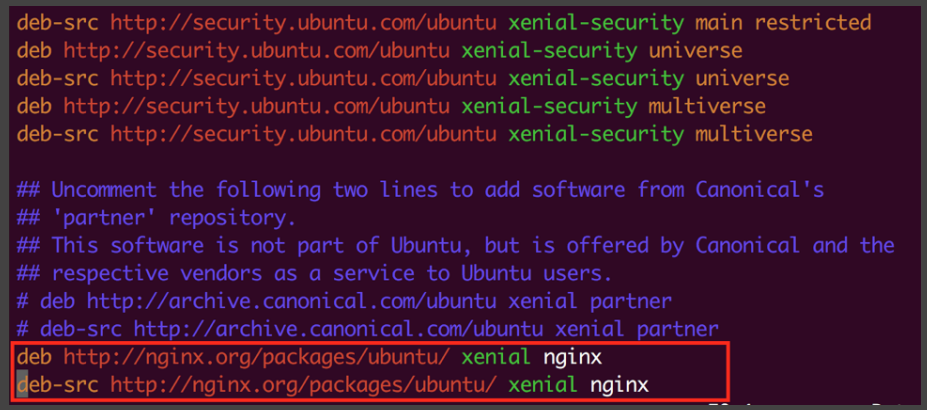
Distribution Codename Reference

Debian:

- 7.x - wheezy
- 6.x - squeeze

Ubuntu:

- 16.04 - xenial
- 14.04 - trusty



Install: `apt-get`

```
sudo apt-get update
sudo apt-get install nginx
```

Check Installation

NGINX will run on port 80 by default

Location of Files

NGINX Executable

```
/usr/sbin/nginx
```

Configuration File

```
/etc/nginx
```

Log Files

```
/var/log/nginx
```

Building NGINX From Source

General Steps:

- Download `.tar` file
- Extract the archive
- Run the `.configure` tool
- Add modules with various parameters using:

```
./configure --<param>=<paramValue>
```

- Run `make && sudo make install`

Command Steps

Copy the Mainline download link here:
http://nginx.org/en/download.html

```
$ sudo wget http://<nginx_mainline_verison>
$ tar -xvf <nginx_mainline_version>.tar.gz
$ sudo apt-get install libpcre3-dev build-essential libssl-dev
$ cd <nginx_mainline_verison>
$ ./configure --with-http_ssl_module --with-debug --with-<other_modules>
$ make && sudo make install
```

Important Notes

Make sure to download requisite libraries PRIOR to compiling the binary

Specify file paths as needed:

- --prefix=path
- --sbin-path=path
- --conf-path=path
- --error-log=path
- --http-log=path

```
# Example
./configure --sbin-path=/usr/local/nginx --error-log=/logs
```

ADDITIONAL RESOURCES

Further Information

- NGINX Documentation
- NGINX Admin Guides
- NGINX Blog

Q&A

- Survey!
- Sales: nginx-inquiries@nginx.com