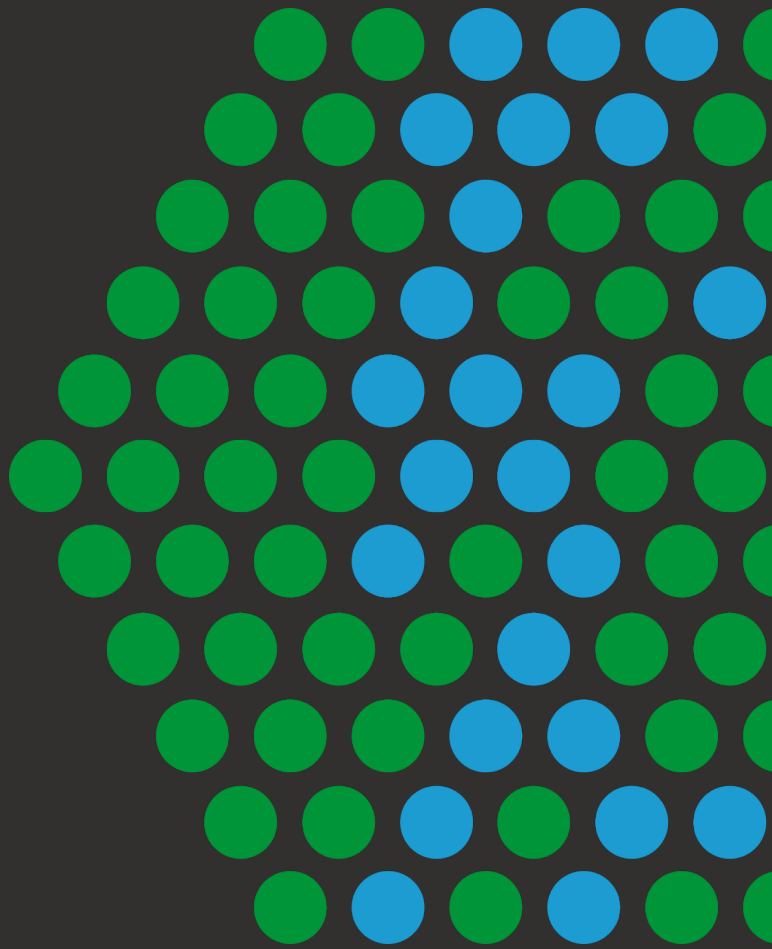
NGINX
Part of F5

NGINX Core



Introduction

- Vicki Shreiner
- NGINX
- Senior Technical Trainer

Prerequisites

- General understanding of web servers
- Familiarity with Linux OS and command line
- Ability to edit files with nano, vi or vim
- Basic knowledge of HTTP and TCP/IP
- Basic understanding of networking

Course Objectives

After completing this course you will be able to:

- Serve static content
- Proxy connections to upstream servers
- Monitor systems using the NGINX dashboard
- Configure and customize NGINX logging
- Use NGINX variables

Course Objectives (cont'd)

- Use variables in rewrites, routing, and maps
- Route connections
- Configure HTTP load balancing
- Configure servers through the NGINX API
- Configure a proxy cache
- Configure HTTPS & site security

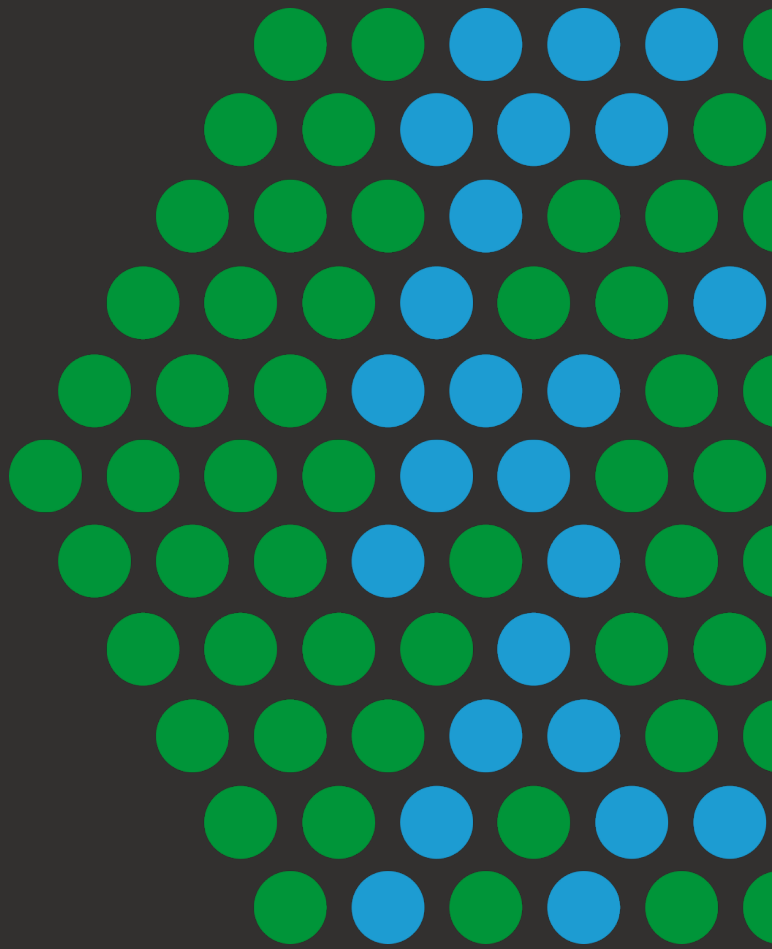
Agenda

- Session 1
  - Introduction to NGINX
  - NGINX Architecture
  - Configuration Contexts
  - Serving Content – Server
  - Serving Content – Location
  - Using Variables
  - Logging
  - Proxying HTTP Requests
- Session 2
  - Routing HTTP Requests
  - Mapping Variables
  - Load Balancing
  - Using the NGINX API
  - Health Checks
  - Caching
  - Encrypting Web Traffic
  - Restricting Access

NGINX
Part of F5

Module 1

Introducing NGINX



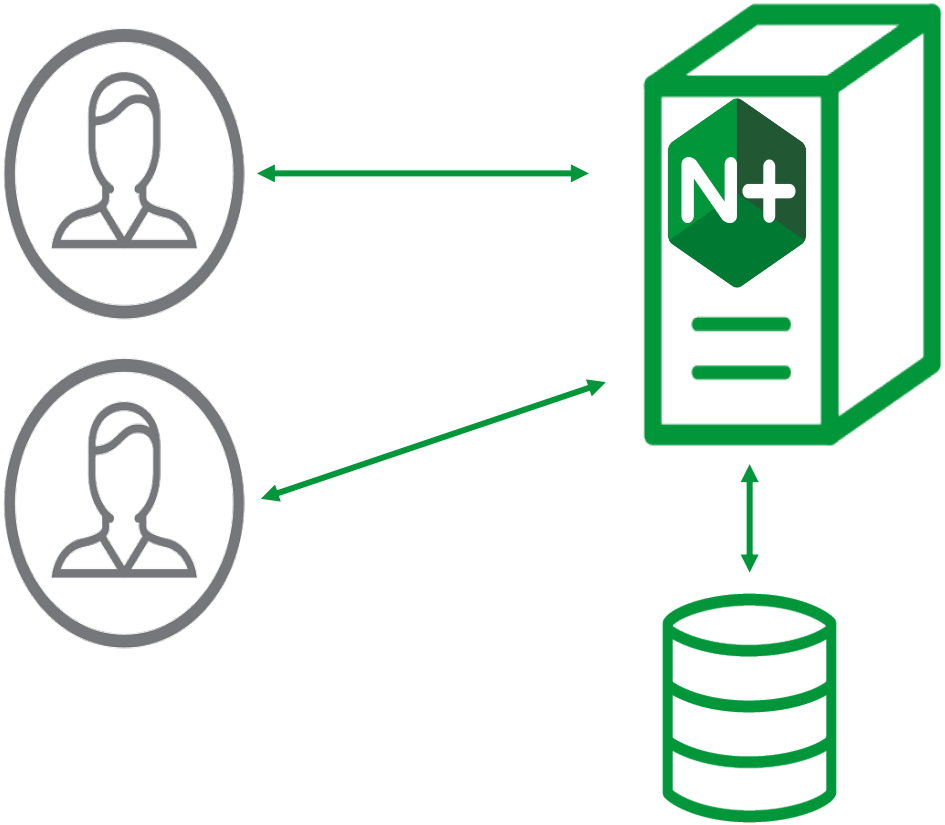
Module Objectives

- Describe most common use cases
- Describe Free/Open Source (F/OSS) features
- Describe NGINX Plus features

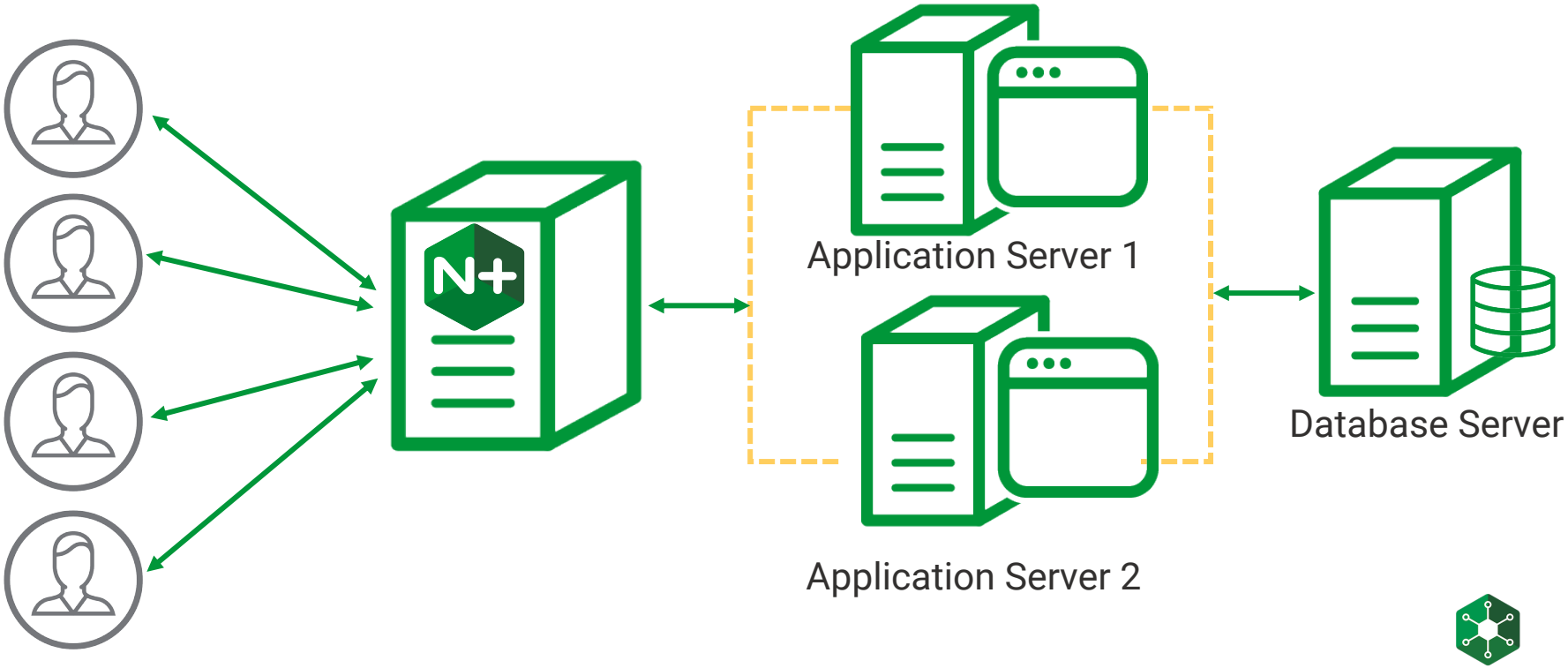
NGINX Application Platform

- NGINX Plus
- NGINX Web Application Firewall (WAF)
- NGINX Unit
- NGINX Controller

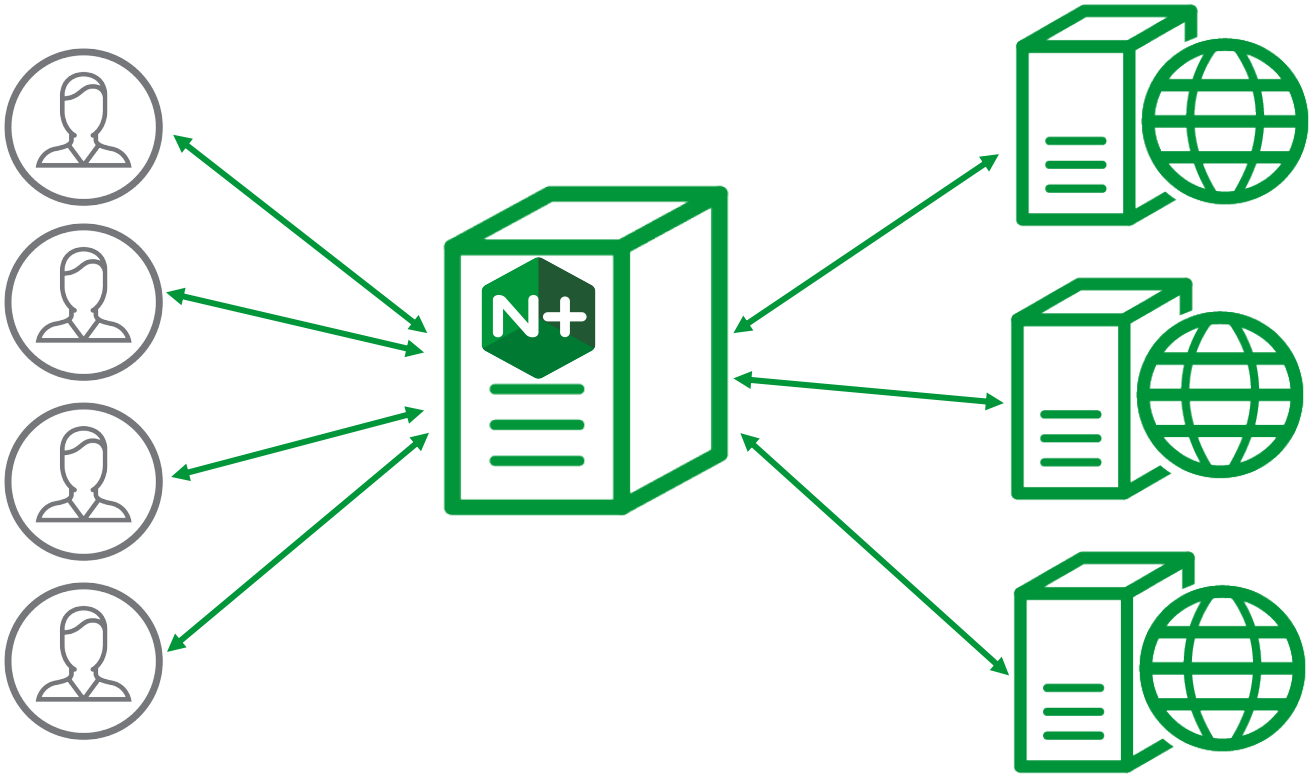
NGINX as a Web Server



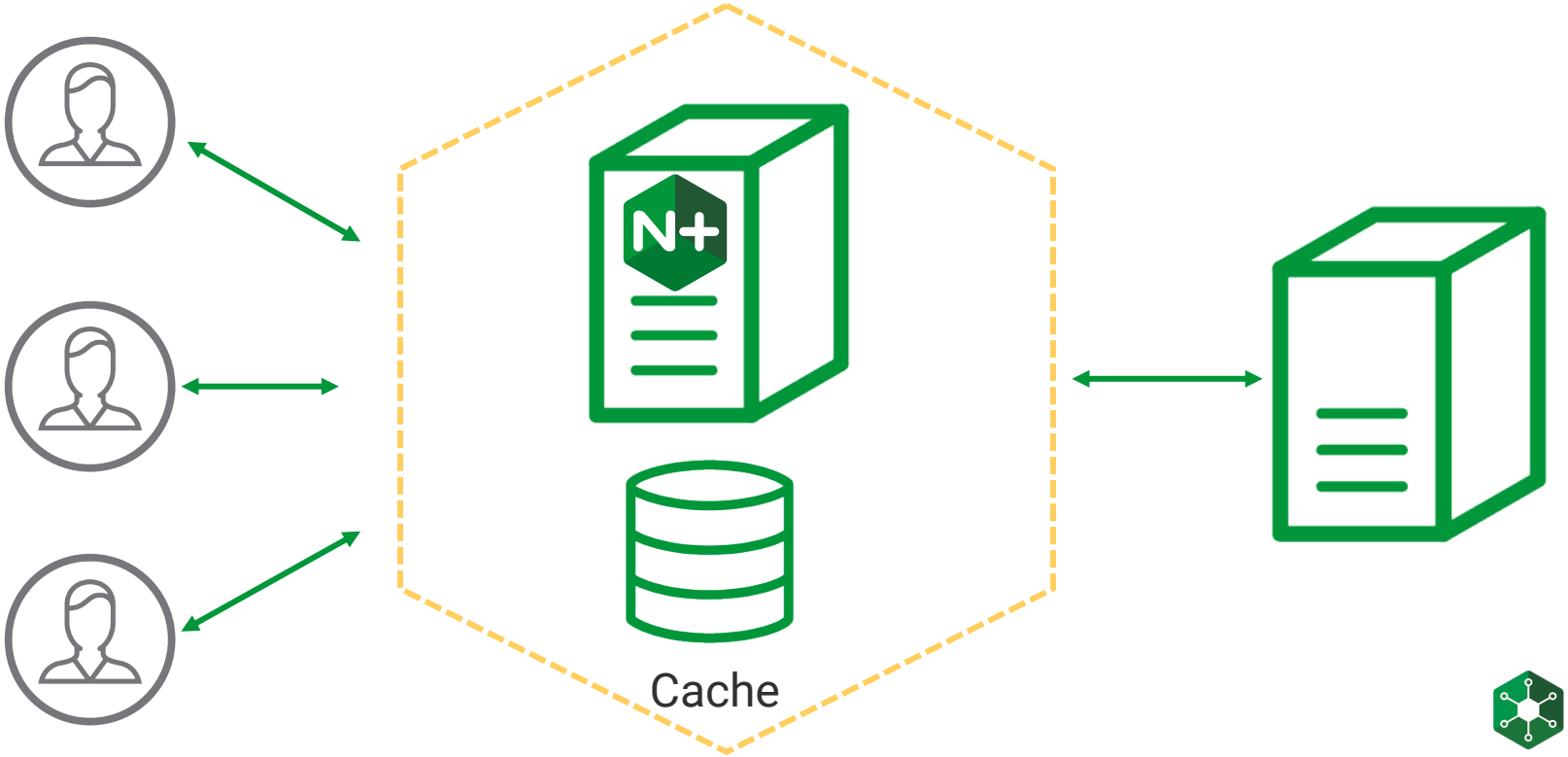
NGINX as a Reverse Proxy Server



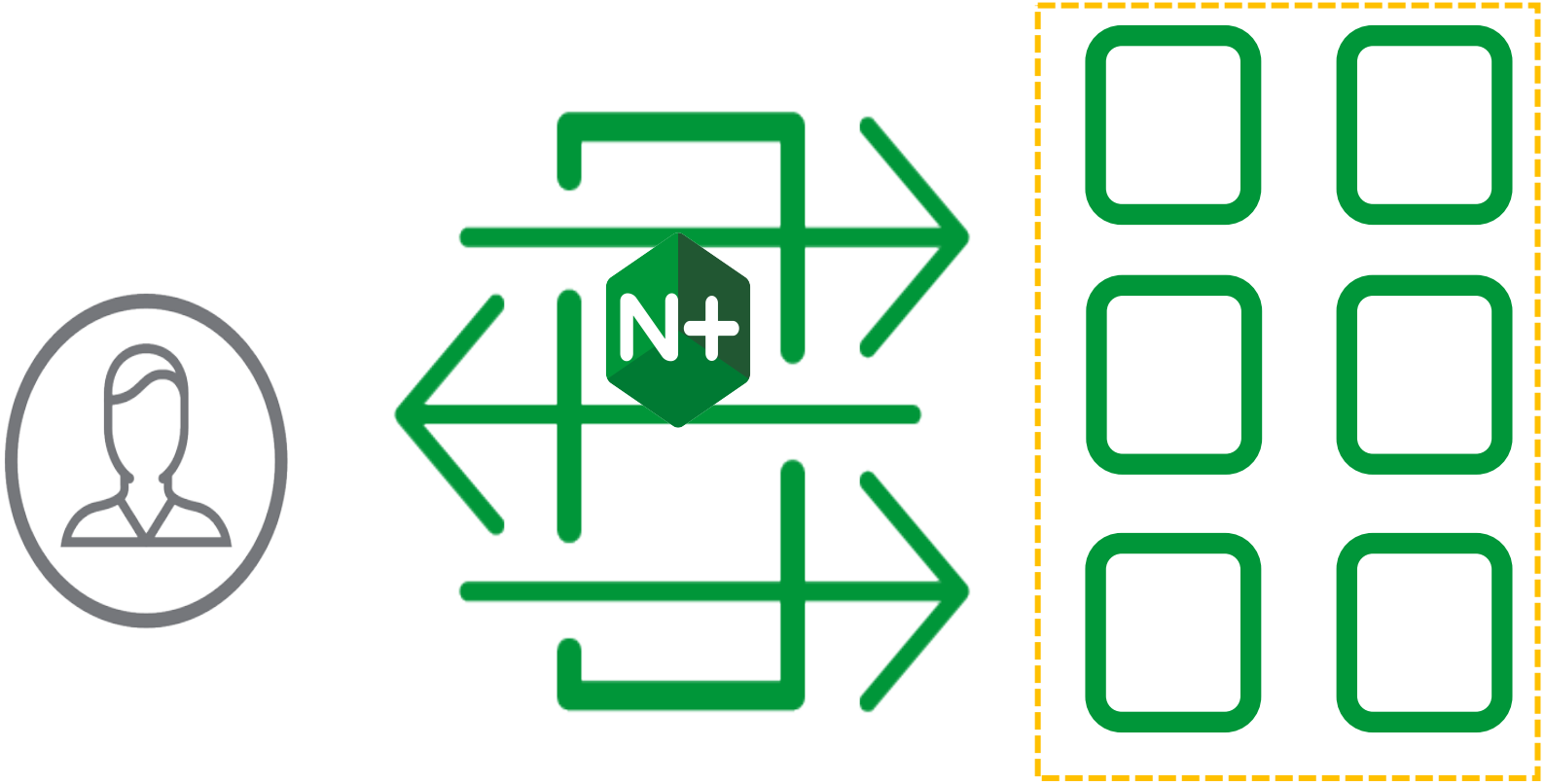
NGINX as a Load Balancer



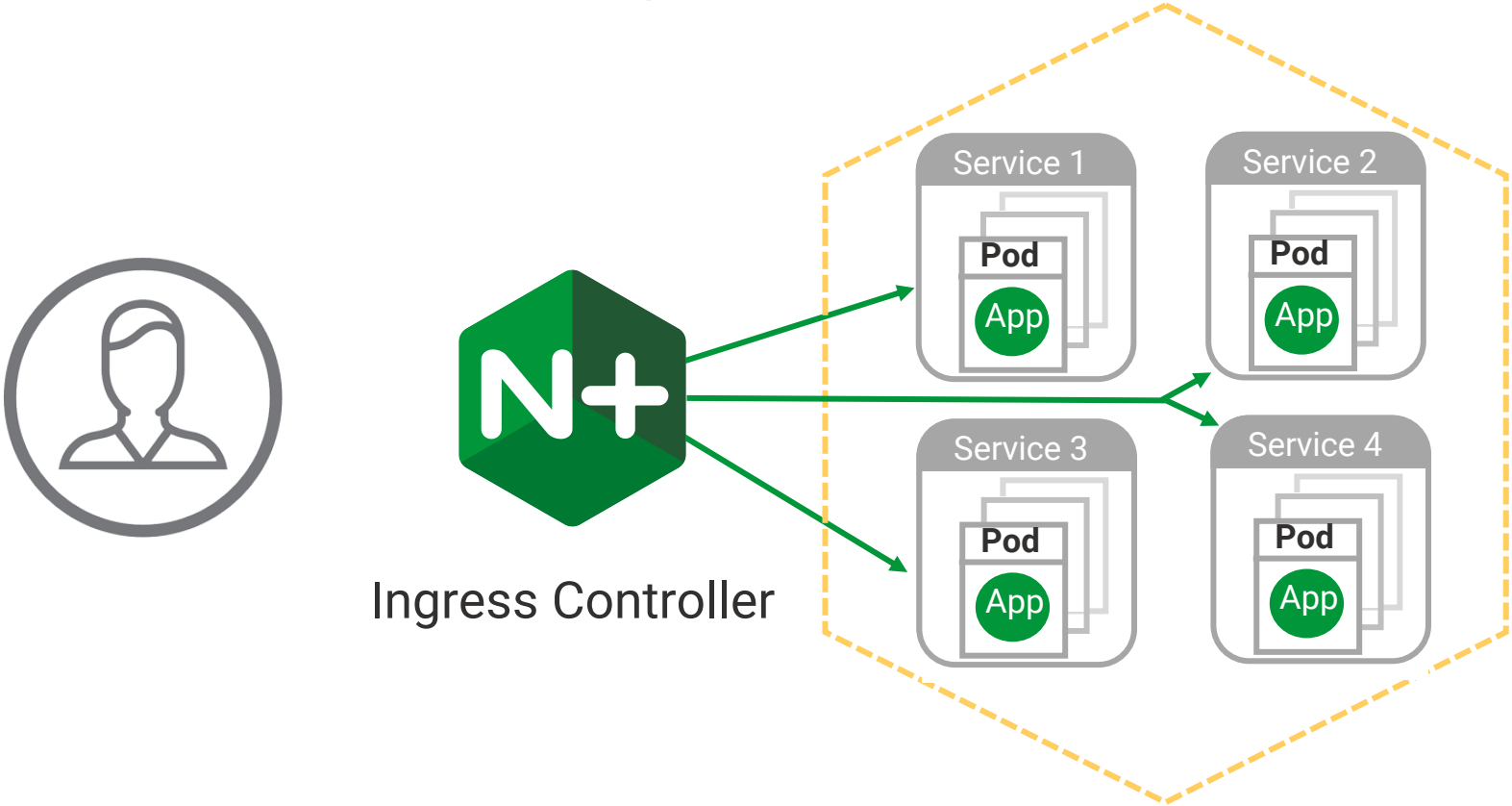
NGINX as a Content Cache



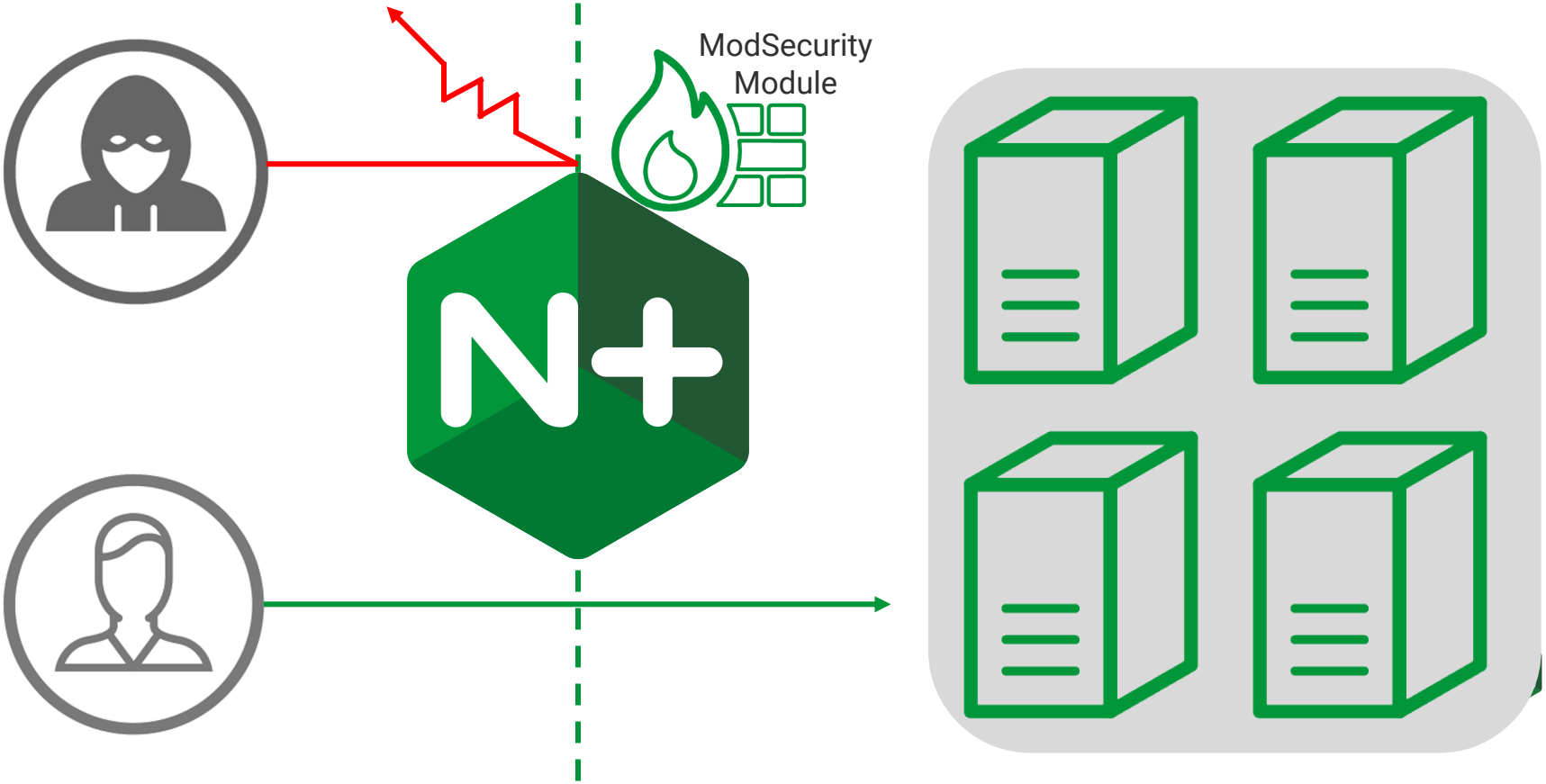
NGINX as an API Gateway



Kubernetes Ingress Controller

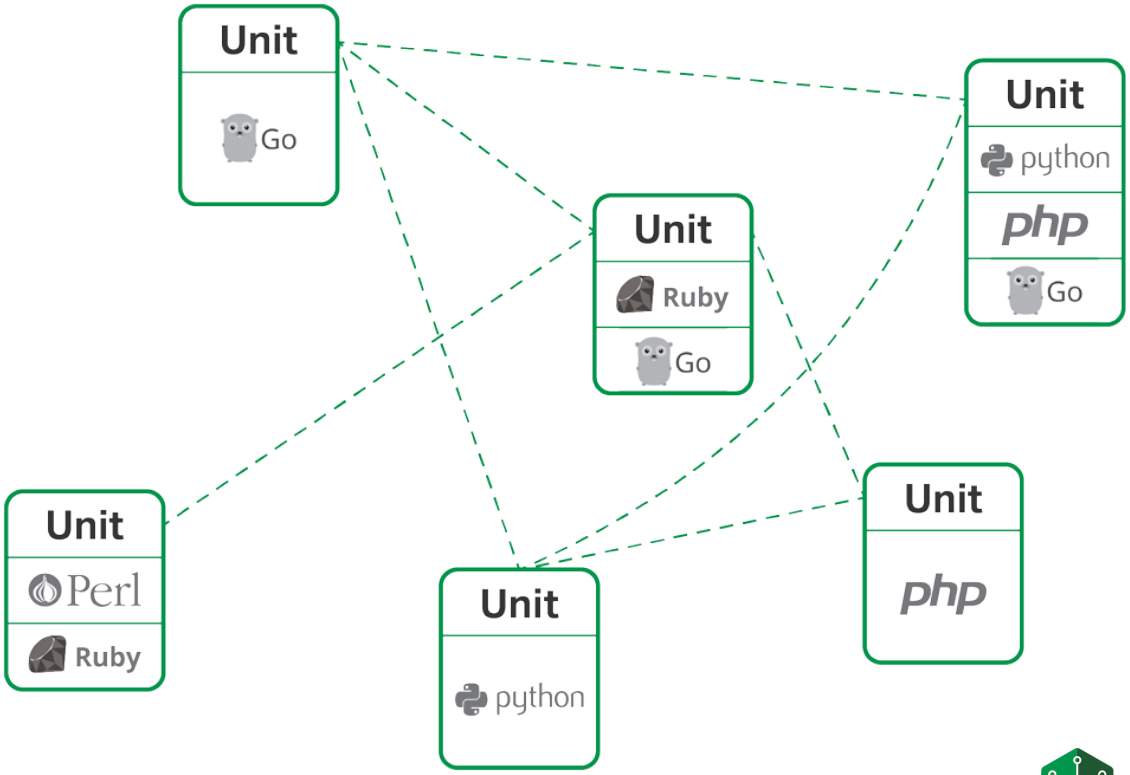


Web Application Firewall

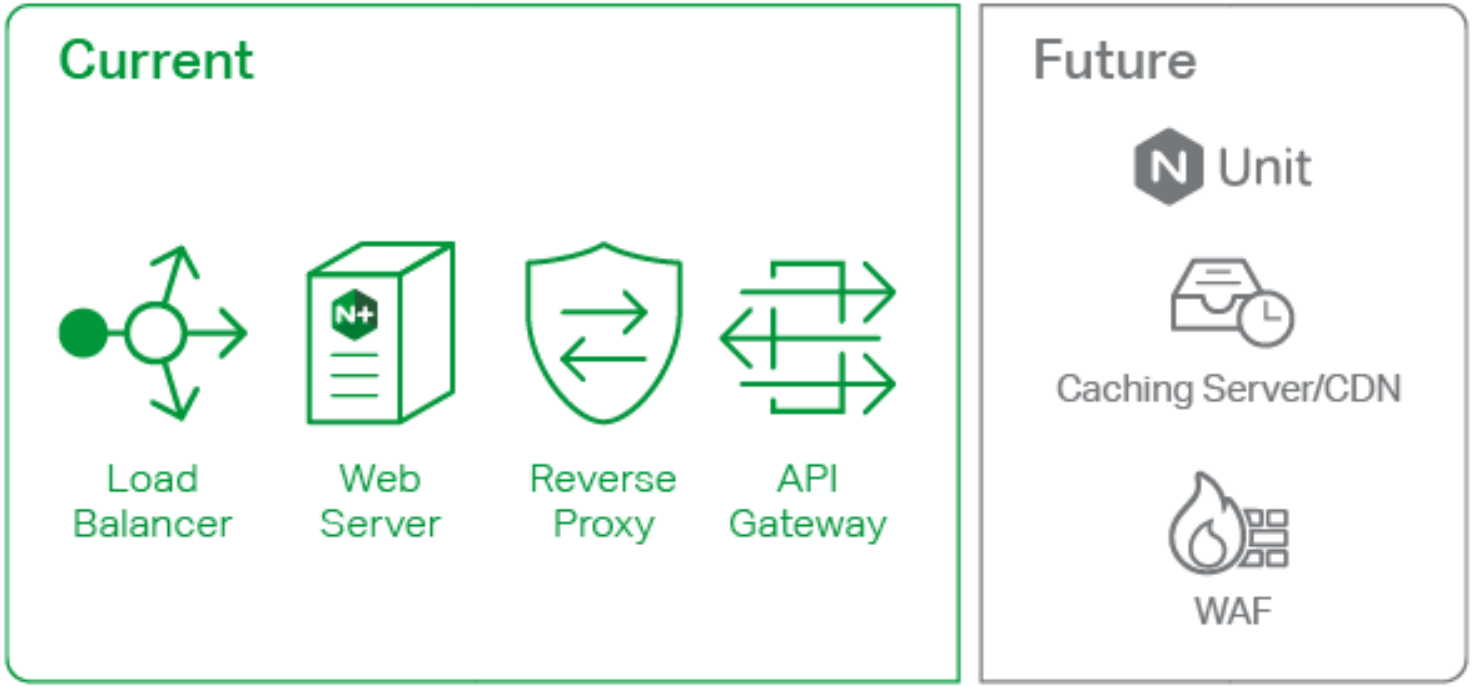


NGINX Unit

- Go
- Perl
- PHP
- Python
- Ruby
- Node.js
- Multiple versions on the same server



NGINX Controller



Open Source and NGINX Plus

Features
Load Balancer (HTTP/TCP/UDP & Layer 7 request routing)
Content Cache (Static/Dynamic)
Web Server/Reverse Proxy
Security Controls/Rate Limiting
Monitoring
Web Application Firewall

https://www.nginx.com/products/nginx/#compare-versions

NGINX Plus Only

Features
Session Persistence
Cache-purging APIs
Active Health Checks
Security – JWT Authentication, OpenID Connect SSO
Extended Status & Metrics
Dynamic Reconfiguration
API Gateway
High Availability
Key-value Store

Knowledge Check

True/False?

NGINX is both a free, open source project and a commercial product.

True

Knowledge Check

NGINX Plus can function as which of the following?

- ☑ A Web content server
- ☑ A reverse proxy server
- ☑ A content cache server
- ☑ An API gateway server
- ☑ An ingress controller
- ☒ An application server
- ☒ A controller server

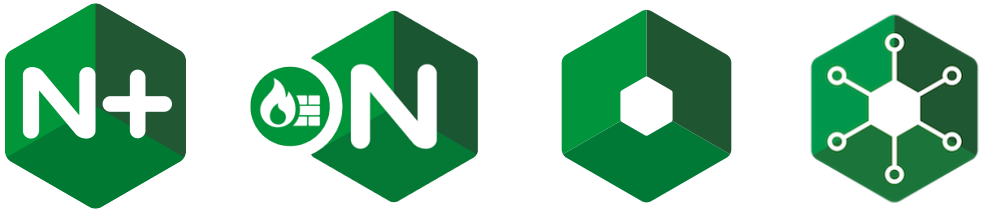
Knowledge Check

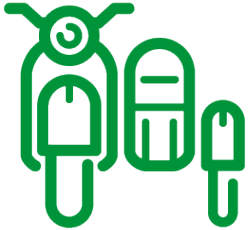
NGINX Web application firewall (WAF):

- ☒ Is included in the NGINX Plus binary.
- ☑ Is a dynamic module built on ModSecurity software.
- ☑ Improves HTTP security through a signature-based rule set.

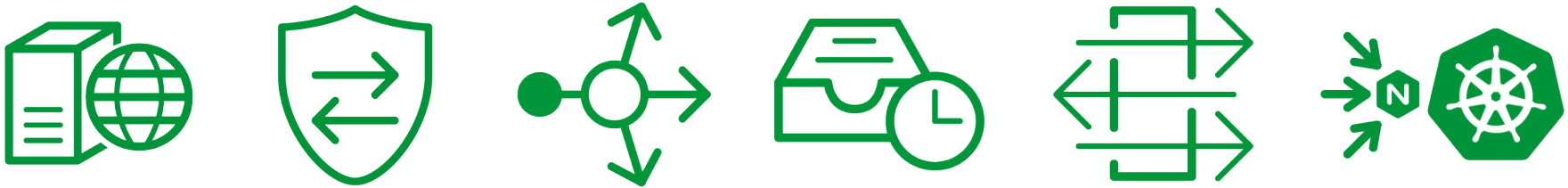
Module Summary

- The NGINX Application Platform consists of
  - NGINX Plus
  - WAF
  - NGINX Unit
  - NGINX Controller





- NGINX Plus can be configured in many ways:



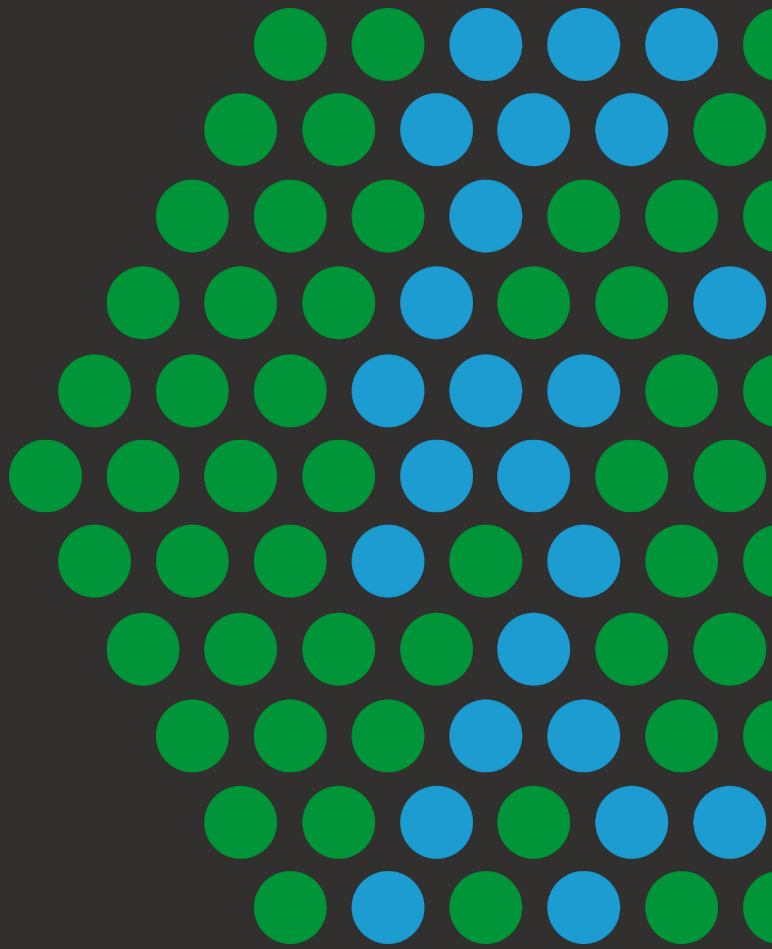
Lab 1 – Log into Ubuntu on AWS

- Log into your Ubuntu instance on AWS
- NGINX Plus is installed

NGINX
Part of F5

Module 2

NGINX Architecture



Module Objectives

- Describe NGINX architecture
- Execute NGINX basic commands

NGINX Plus Architecture

- Processes
  - Master process
  - Child processes
    - Cache manager
    - Cache loader
    - Worker processes
- Non-blocking I/O
- NGINX Reload

Master Process

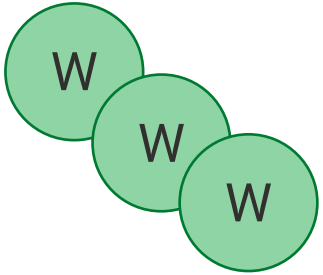
```
$ ps aux | grep nginx
root      11650  0.0  0.3  32936  3276 ?        Ss   22:21   0:00 nginx: master process nginx
nginx     11749  0.0  0.2  33332  2784 ?        S    22:26   0:00 nginx: worker process
```

```
$ head /etc/nginx/nginx.conf
user  nginx;
worker_processes  auto;

error_log  /var/log/nginx/error.log notice;
pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
}
```

```
$ cat /var/run/nginx.pid
11650
```

Child Processes

- Cache manager
- Cache loader
- Worker processes



Non-blocking I/O

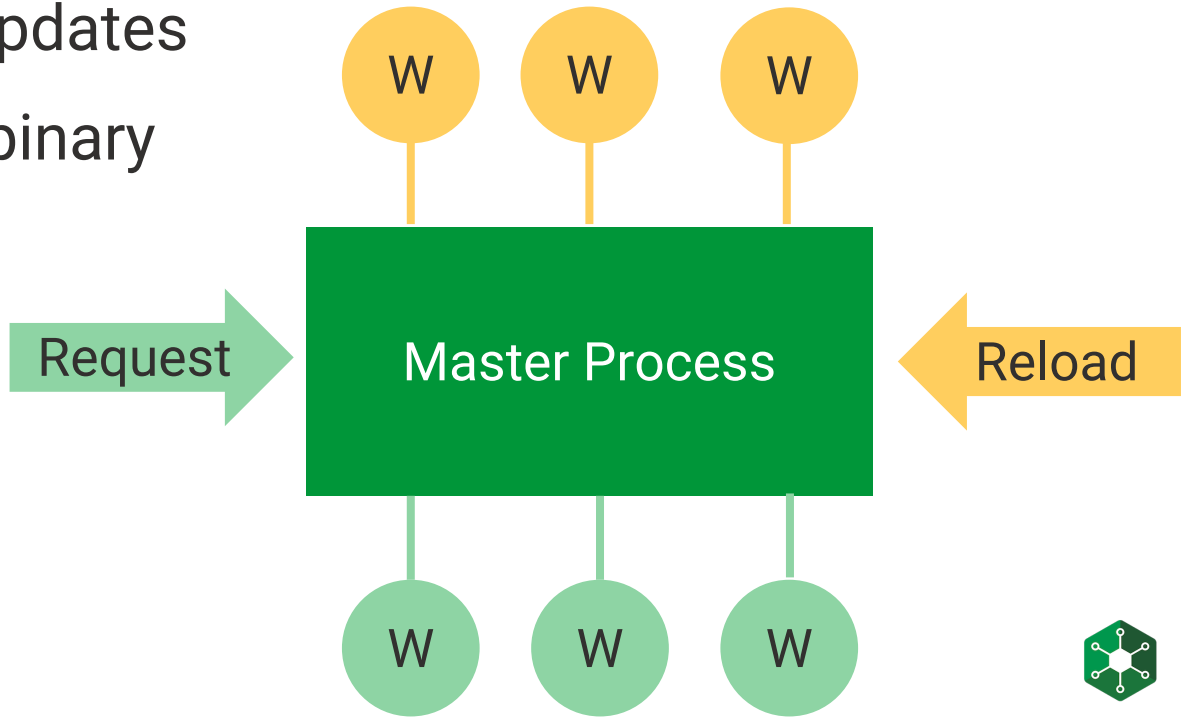
Blocking

Non-blocking

Time

No Downtime

- Configuration updates
- Update NGINX binary



NGINX Commands

- Check NGINX version

```
$ nginx -v
```

- Check config file syntax

```
$ sudo nginx -t
```

- Display current configuration(s) and check file syntax

```
$ sudo nginx -T
```

- Reload Configurations

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

NGINX Resources

- Beginners Guide
  http://nginx.org/en/docs/beginners_guide.html
- Documentation
  http://nginx.org/en/docs
- Complete NGINX Cookbook
  https://www.nginx.com/resources
- NGINX Blogs
  https://www.nginx.com/blog
- Directive “man page”
  http://nginx.org/r/*directive*

Knowledge Check

True/False?

To reconfigure NGINX you must restart it.

False

Knowledge Check

True/False?

NGINX has a non-blocking I/O architecture.

True

Knowledge Check

Which of the following NGINX commands checks the configuration file syntax?

- ☑ `nginx -t`
- ☑ `nginx -T`
- ☒ `nginx -v`
- ☑ `nginx —s reload`
- ☒ `ps aux | grep nginx`

Summary

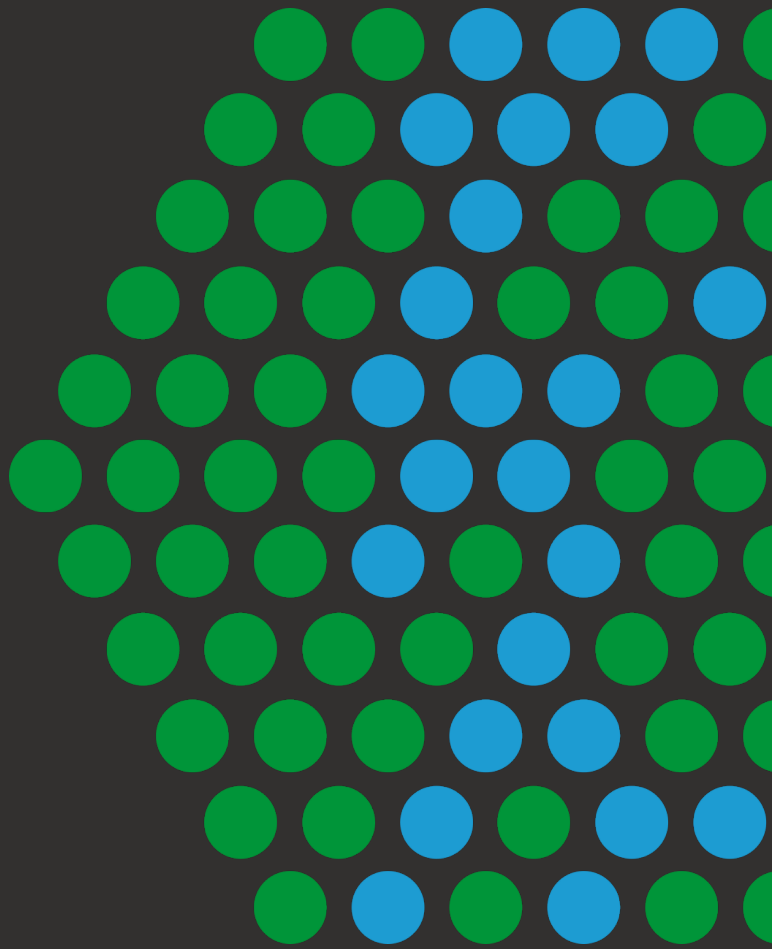
- NGINX uses non-blocking I/O architecture
- NGINX tracks its master process via a process ID
  - Changes when NGINX is restarted
- NGINX does not stop processing during
  - Configuration updates (nginx –s reload)
  - Binary updates
- NGINX reloads can be done safely in production

Lab 2 – View NGINX Processes

NGINX
Part of F5

Module 3

Configuration Contexts



Module 3 Objectives

- Locate NGINX configuration file(s)
- Describe the role of contexts, blocks, and directives in your configuration file(s)

Configuration File Locations

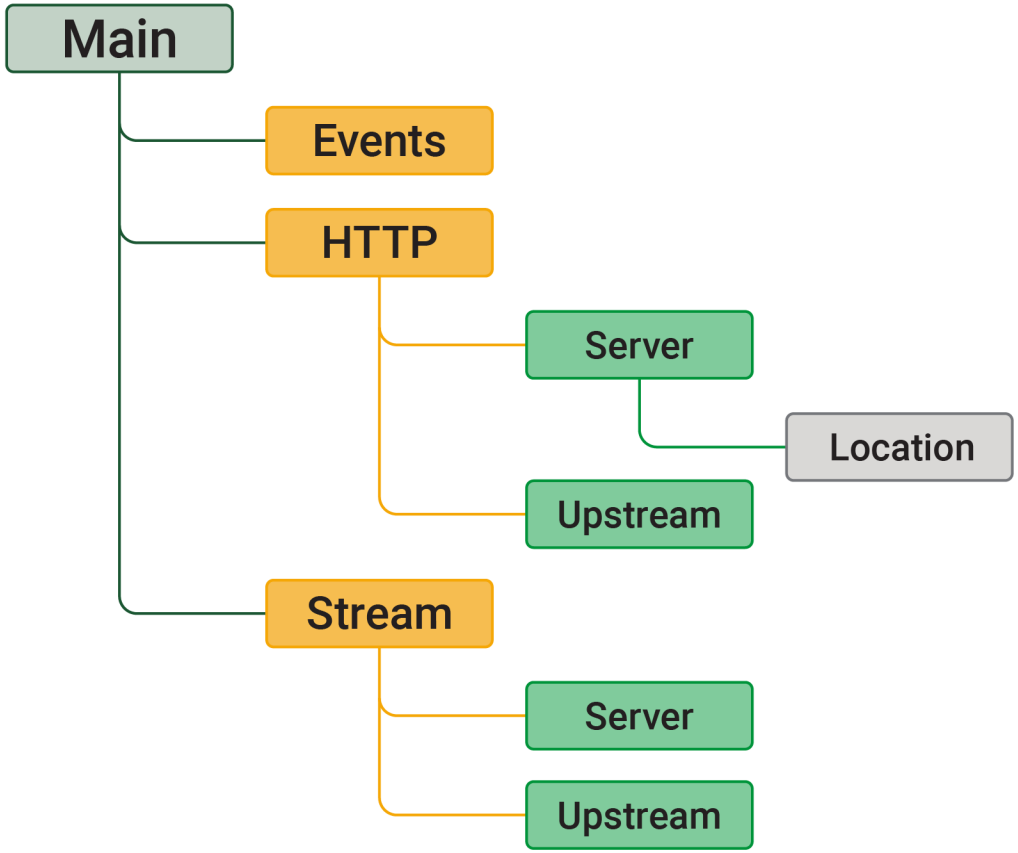
- Main configuration file

  /etc/nginx/nginx.conf

- Default configuration file

  /etc/nginx/conf.d/default.conf

Configuration Contexts



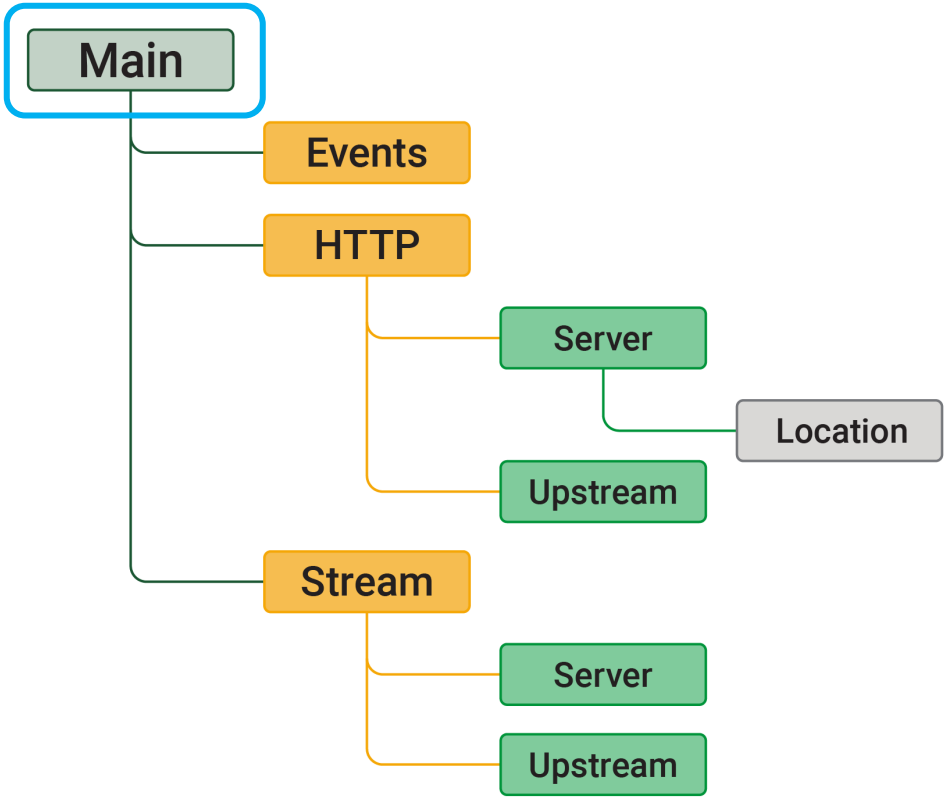
NGINX Config has

- One main context
- One HTTP context
- Our focus
  - **Server** context
  - **Location** context

Link Back

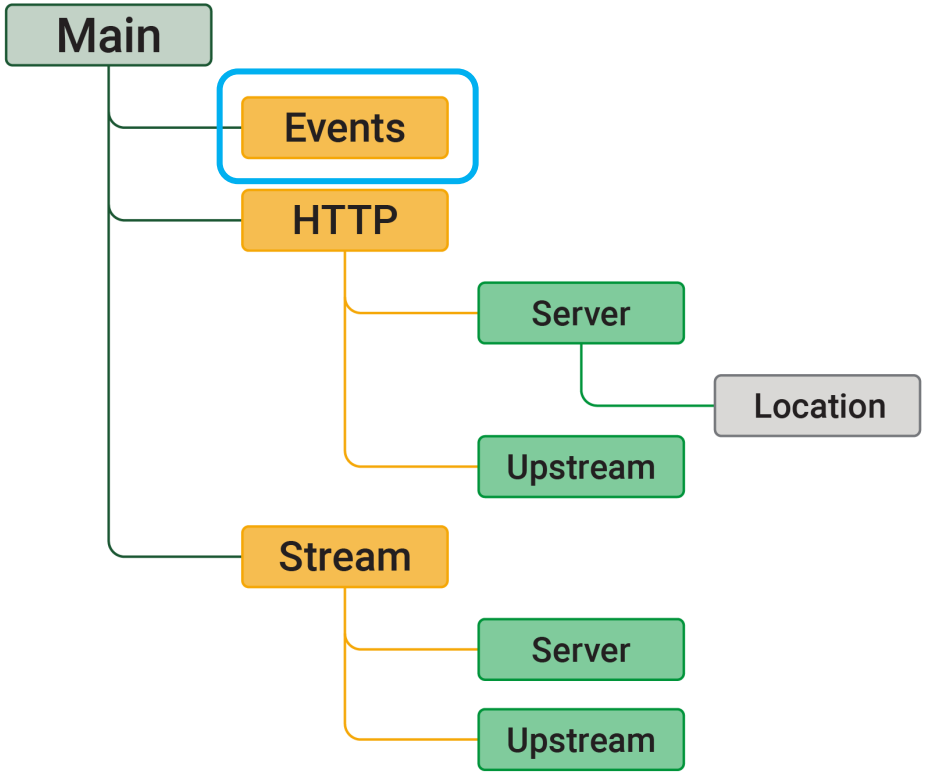
Main Context

- Highest level directives
- Examples
  - Number of worker processes
  - Linux username for NGINX
  - File locations:
    - Error log
    - Process ID (PID)



Events Context

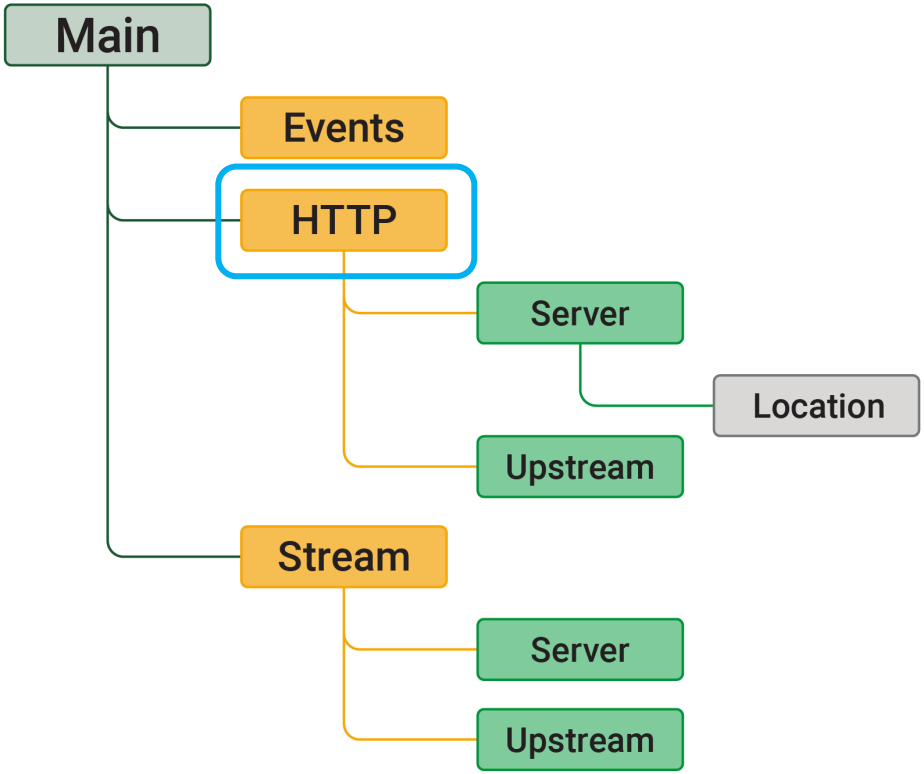
- Manages connection processing directives
- Example:
  - Number of connections per worker process
- We use defaults



HTTP Context

Determines how NGINX handles HTTP and HTTPS connections

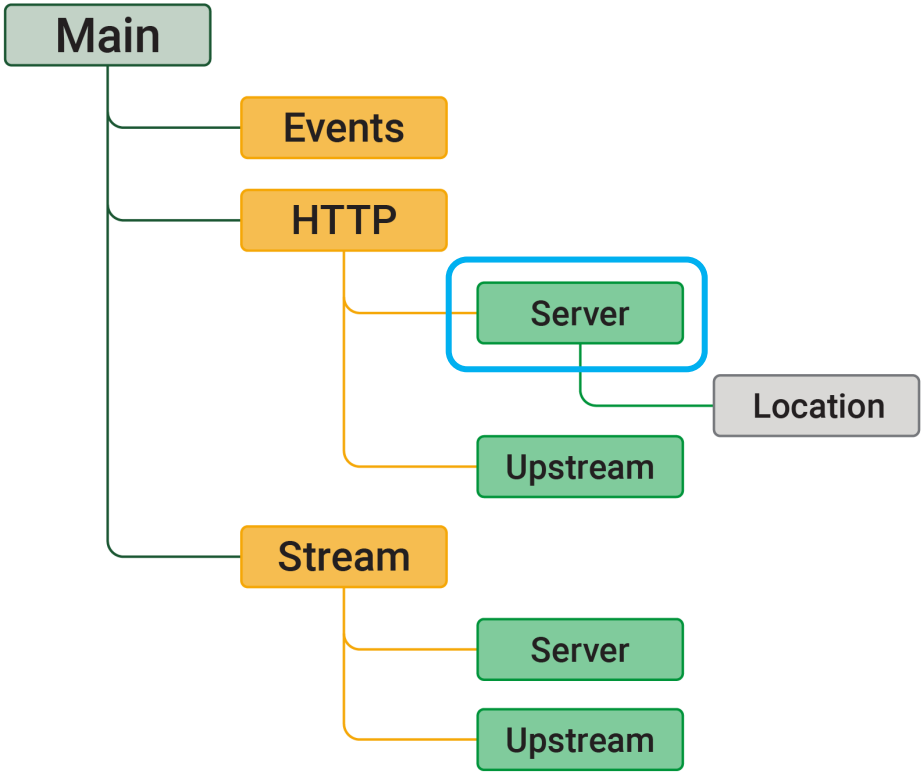
- log formats/locations
- include statement



Server Context

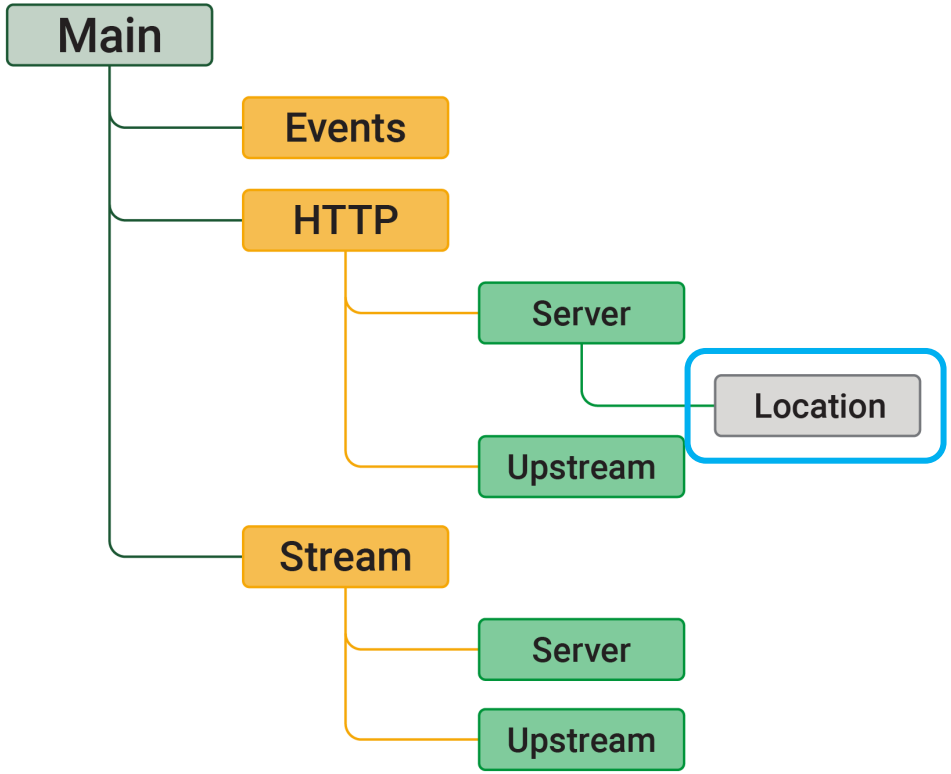
Defines virtual server that responds to a request for

- Domain name
- IP address
- Unix socket

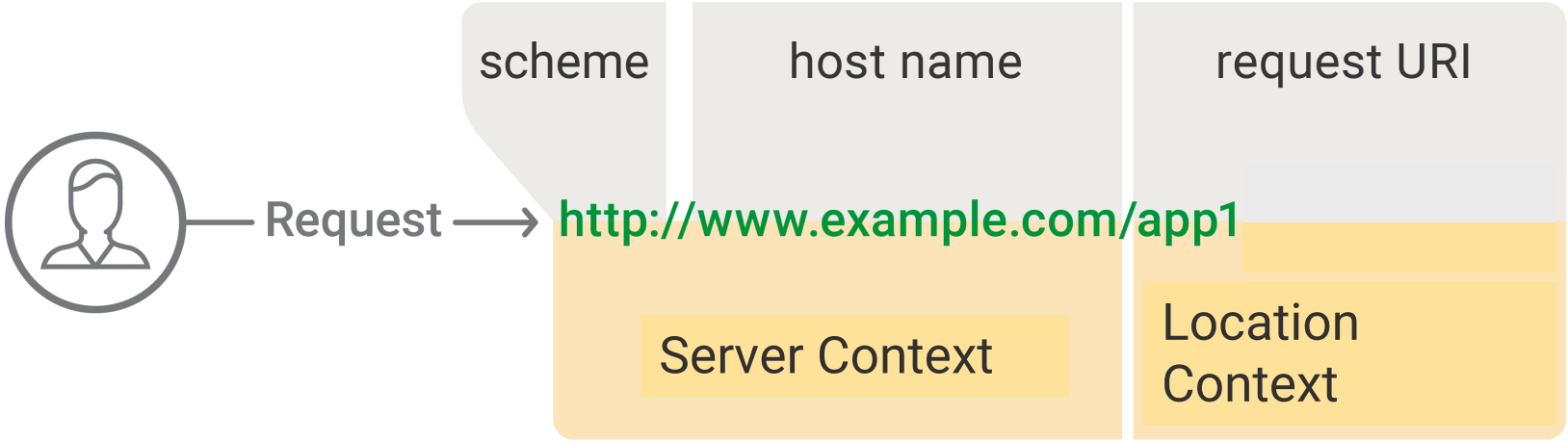


Location Context

- Further specifies how to process an HTTP request
  - Based on request URI
  - Filesystem path
  - Perform request string matching

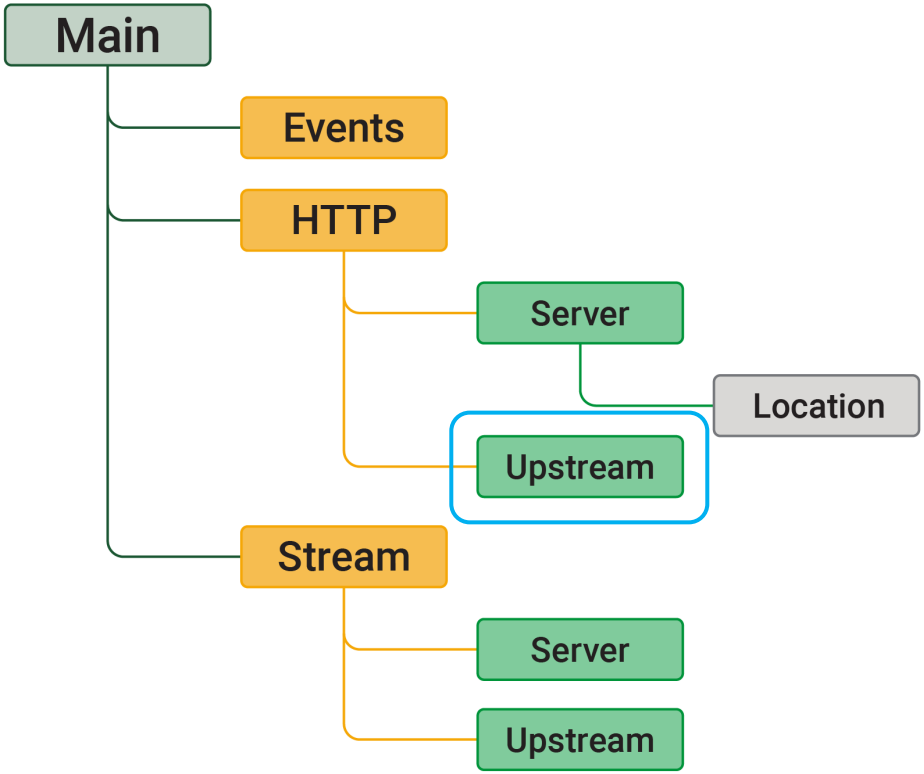


Server & Location Contexts



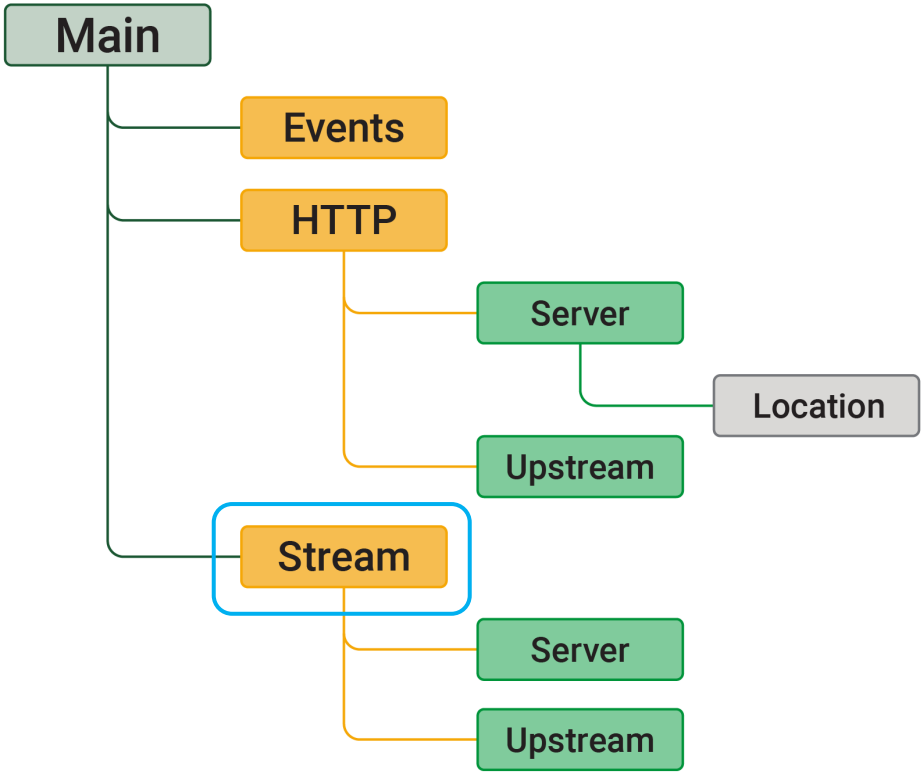
Upstream Context

- Defines group of backend servers
  - Application servers
  - Web servers
  - Other

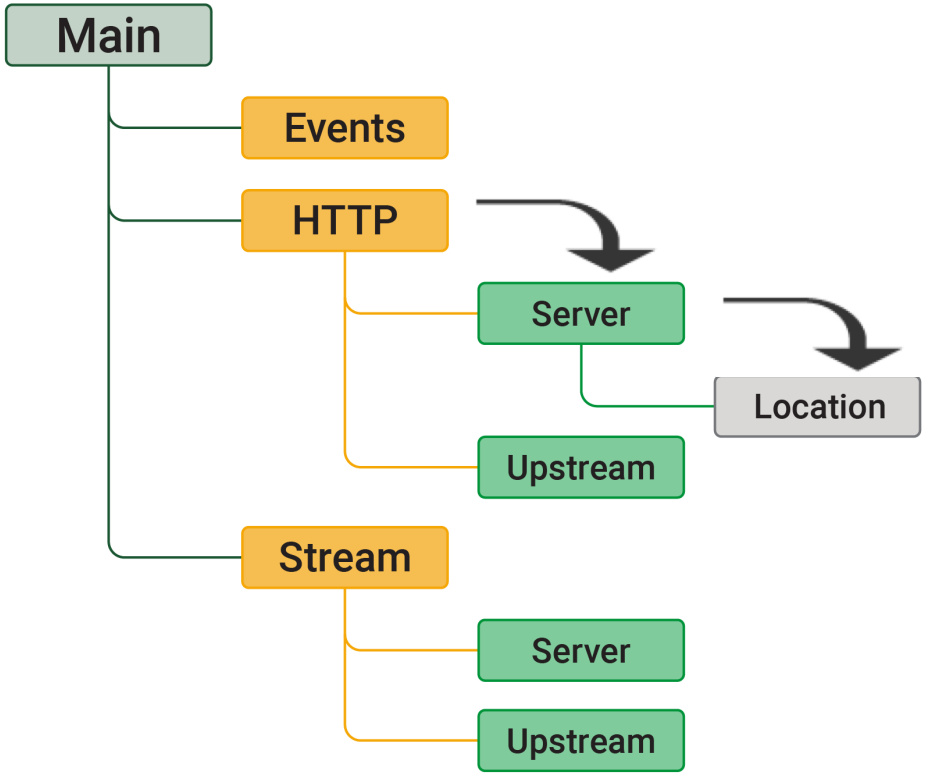


Stream Context

- Defines handling of layer 4 network traffic
  - UDP
  - TCP



Inheritance



Directives

- Directive: Single-line statement that controls behavior
  - ends in a semicolon
- Block – group of directives
  - encased in curly braces

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

Server Block Start

Listen Directive

Root Directive

End of Server Block

Directive Parameters

Examples of listen directive parameters:

```
listen 443 ssl;
#Defines an HTTPS server on port 443

listen 443 ssl http2;
#Allows HTTP/2 protocol for client HTTP requests

listen 53 udp;
#Defines UDP protocol for all port 53 traffic
```

Directive Types

Type	Example
Standard	listen, root
Array	log_format
Command	proxy_pass

Standard Directive Example

```
http {
    index index.html;
     server {
         listen 80;
         root /usr/home/share/nginx/html;
    }
    server {
         listen 8080;
         root /test/home/html;
     }
}
```

Array Directive Example

```
http {
    log_format custom '"Request: $request_uri\n Status
         $status\n Backend Address $upstream_addr"';
    server {
        listen 80;
        access_log /var/log/nginx/server1.log custom;
    }
 }
```

Command Directive Example

```
server {
    listen 80;
    root /usr/home/share/nginx/html;
    access_log /var/log/nginx/requests.log combined;
    proxy_pass http://backend-app-server;
}
```

Directive: root

Filesystem pathname to use for files served

Syntax: **`root path;`**

```
server {
    listen 80;
    root /project/share/html;
}
```

Directive: index

Filename to look for in the specified order and will be appended to the root path of file returned

Syntax: ***index file1 file2 … /absolute_path/file;***

```
server {
    listen 80;
    root /project/share/html;
    index fun.html;
}
```

The include Directive

```
http {
    ...
    include /etc/nginx/conf.d/*.conf
}
```

Module Summary

- Configuration files hold NGINX directives
- Directives are statements that control nginx behavior
- We work mainly in HTTP, Server, and Location contexts
- Child configuration (blocks) inherit parent configurations unless more specifically directed

Knowledge Check

In which of the following contexts is the domain name or IP address configured?

- ☒ Main context
- ☒ HTTP context
- ☑ Server context
- ☒ Location Context

Knowledge Check

In which of the following commands checks the configuration files' syntax?

- ☑ `nginx -T`
- ☑ `nginx -t`
- ☒ `nginx -s`
- ☒ `nginx -S`

Knowledge Check

In which of the following contexts is the address of a backend pool of servers configured?

- ☒ Main context
- ☑ upstream context
- ☒ Server context
- ☒ Location Context

Knowledge Check

NGINX directives may

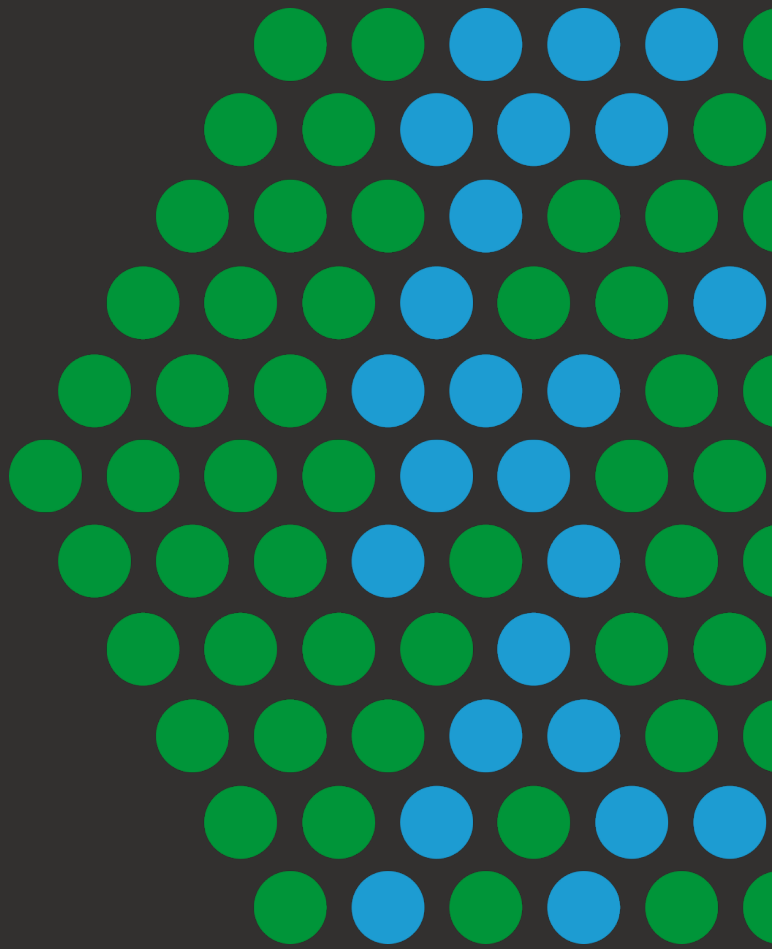
- ☑ tell NGINX how to process HTTP requests.
- ☑ indicate to which IP and port to respond.
- ☑ indicate to which protocol to respond.
- ☑ all of the above.

Lab 3 – Serving Pages

NGINX
Part of F5

Module 4

Serving Content: Server

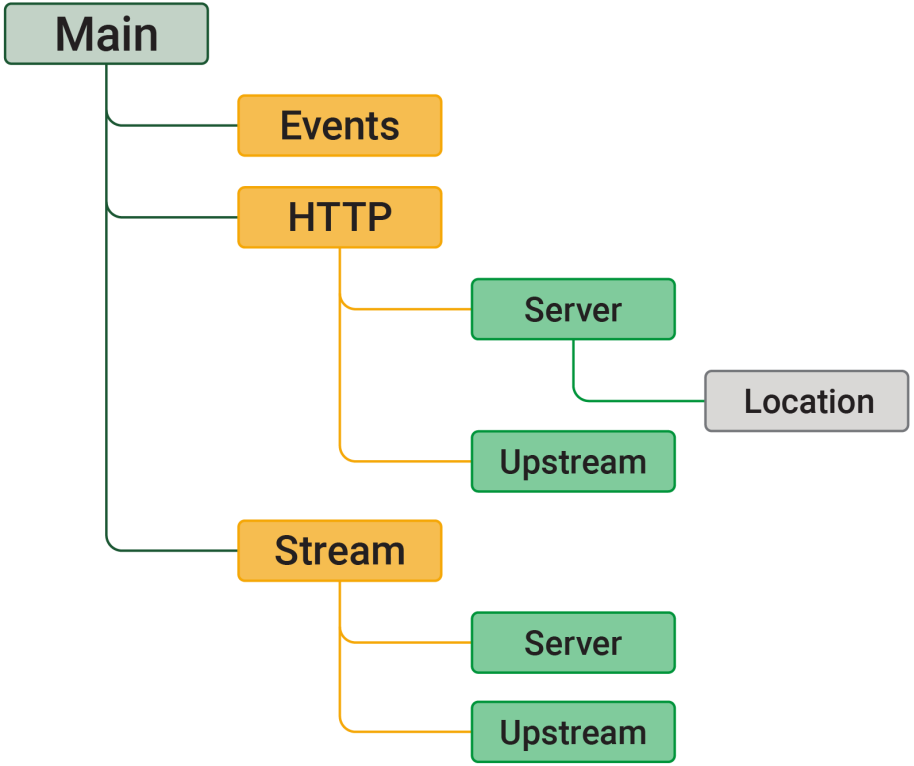


Module 4 Objectives

- Identify the server block that responds to a request
- Configure a simple web server that serves static pages and images

Serving Content

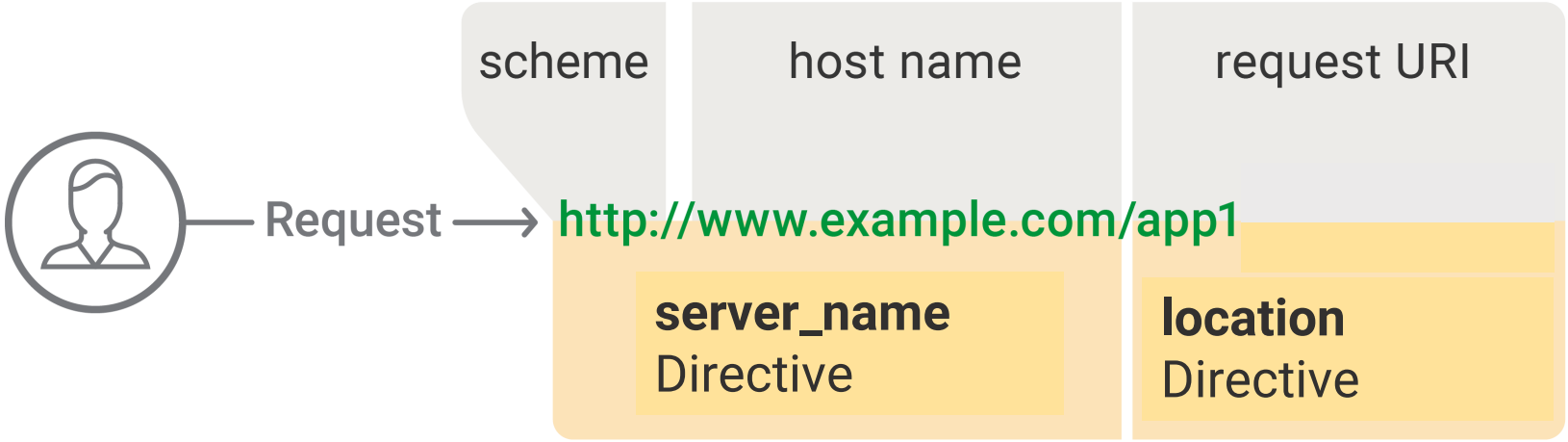
- `http` – high level processing
- `server` – virtual server handling requests
- `location` – based on request URI



NGINX Defaults

```
server {
    listen 0.0.0.0:80;
    server_name “”;
    root /usr/share/nginx/html;
    index index.html;
}
```

Breaking Down a Request



Server Selection

```
server {
    listen 127.0.0.1:80;
    server_name example.com;
    root /usr/share/nginx/html;
}
```

The Listen Directive

DIRECTIVE	NGINX USES
`<no listen directive>`	0.0.0.0:80
`listen 127.0.0.1:8080;`	127.0.0.1:8080
`listen 127.0.0.2;`	127.0.0.2:80
`listen 80;`	0.0.0.0:80
`listen unix:/var/run/nginx.sock;`	Must use absolute path

Note: NGINX also listens on IPv6

Which server block responds?

example.com IP is 192.168.1.10

1. http://example.com:8080
2. http://192.168.1.10/
3. http://192.168.1.10:80/
4. http://example.com/

```
server {
    listen 192.168.1.10:8080;
}

server {
    listen 192.168.1.10;
}

server {
    listen 80;
}
```

server_name Directive

Request: http://www.example.com

```
server {
     listen 192.168.1.10;
     server_name www.example.org;
      ...
}
server {
     listen 192.168.1.10;
     server_name www.example.com;
      ...
}
```

server_name Selection Order

1. exact string match
2. matches with leading wild card
3. matches with a trailing wild card
4. the first regex match

server_name Example 1

Request is host1.example.com

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

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

server_name Example 2

Request is www.example.org

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

server_name Example 3

Request is www.example.com

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

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

default_server Parameter

```
server {
      listen 192.168.1.1;
      server_name www.example.*;
}
server {
      listen 192.168.1.1;
      server_name *.example.net;
}
server{
      listen 192.168.1.1;
      server_name ~^(host1|host2).*\.example\.com$;
}
server {
      listen 192.168.1.1 default_server;
      error_page 404 /40x.html;
}
```

Bad Requests

- Prevent bad requests with server_name
- Close TCP connection with 444 response code

```
server {
    listen 80 default_server;
    server_name nginx.com;
}

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

Module Summary

- listen directive defines IP:Port for virtual servers
- NGINX fills in missing parameters for listen directive
  - Listen directive with IP only wins over Port only
- server_name directive examines host header, differentiates when multiple matching IP:Port servers
  - server_name matches longest matching server name
- default_server used when NGINX can’t match based on listen and/or server_name

Knowledge Check

www.sub.example.com resolves to 192.168.1.1

Which of the following serves the client response?

```
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$; }
```

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

Knowledge Check

www.example.net resolves to 192.168.1.1

Which of the following serves the client response?

```
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$; }
```

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

Knowledge Check

www.training.net resolves to 192.168.1.1

Which of the following serves the client response?

```
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$; }
```

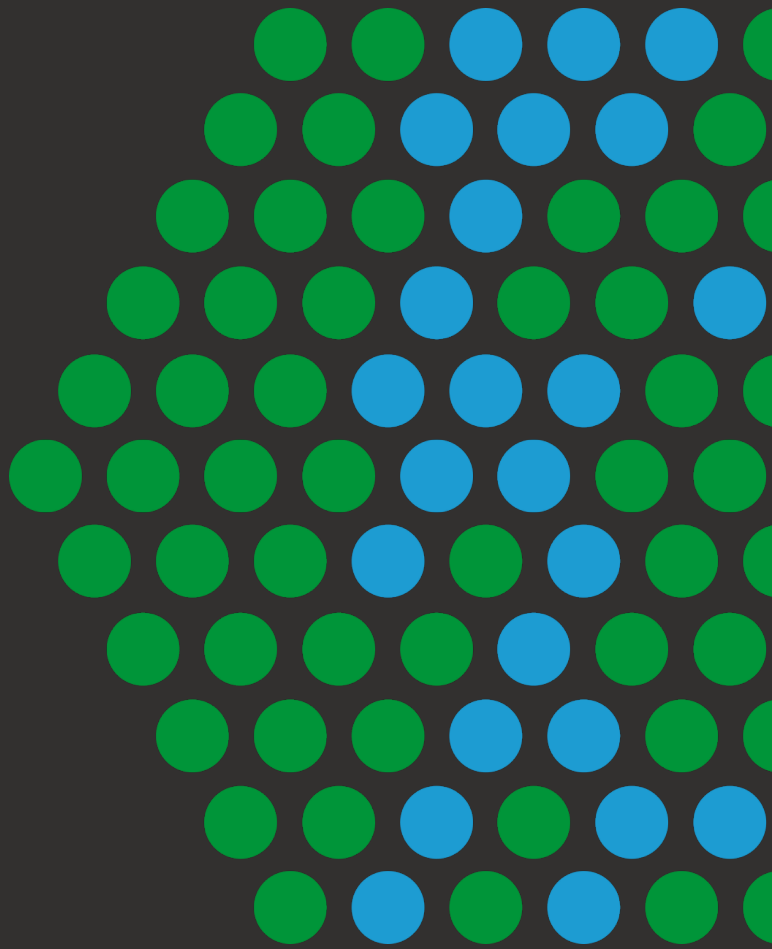
```
server { listen 192.168.1.1:80 default_server;
server_name www.example.org; }
```

Lab 4 – Server Selection

NGINX
Part of F5

Module 5

Serving Content: Location



Module 5 Objectives

- Identify **location** block types
- Identify **location** processing rules
- Configure a web server that serves static content

Location Blocks

```
location /app1 {
}
```

Request → http://www.example.com/app1

scheme | host name | request URI

server_name Directive

location Directive

Location Block types

- Prefix

```
server {
     listen 192.168.1.1;
          location /app1 {…}
          location /parts {…}
          location /shop {…}
}
```

- Regex

```
server {
     listen 192.168.1.2;
          location ~* ^/app(\d+)$ {
     }
}
```

Common Location Modifiers

Modifier	Usage
~*	Case insensitive regex
~	Case sensitive regex
^~	Prevent regex matching

Note: For a full list of location modifiers, refer to the documentation here: http://nginx.org/en/docs/http/ngx_http_core_module.html#location

Location Processing

1. NGINX checks all **prefix** locations
   - Continues processing
2. NGINX checks **regex** locations in order listed
   - Uses **first** match, stops processing

Prefix Locations

- NGINX selects the **longest matching prefix**
- Order doesn’t matter

Request: http://example.com/gallery/images/cat.png

```
location /gallery {
}

location /gallery/images {
}
```

Regex Location

Request: https://example.com/gallery/images/cat.png

```
server {   ...
     location /gallery {
     }
     location ~* ^.*\.(png|jpg)$ {
     }
}
```

Note: Best practice is to list prefix locations first, then regex

Prefix versus Regex Location

Request: https://example.com/gallery/images/cat.png

```
server {    ...
     location /gallery {
     }
     location ~* ^.*\.(png|jpg)$ {
     }
     location ^~ /gallery/images {
     }
}
```

Note: Best practice is to list prefix locations first, then regex

Location Processing Logic - Ex1

Request: http://www.example.com/app1/index.php

```
server {
     listen 80 default_server;
     root /home/ubuntu/web_server_test;

     location / {
     }
     location ~* \.(php|html)$ {
     }
     location ^~ /app1 {
     }
}
```

Location Processing Logic - Ex2

Request: http://www.example.com/app1/index.php

```
server {
     listen 80 default_server;
     root /home/ubuntu/web_server_test;

     location = / {
     }
     location ~* ^.*\.(php|html)$ {
     }
     location /app1 {
     }
}
```

Location Processing Logic - Ex3

Request: http://www.example.com/app1/index.php

```
server {
     listen 80 default_server;
     root /home/ubuntu/web_server_test;

     location / {
     }
     location /app1/ {
     }
     location /app1 {
     }
}
```

Named Locations

```
location /test {
    error_page 404 = @error_fallback;
}

location @error_fallback {
    proxy_pass http://backend1/test;
}
```

Module Summary

- NGINX selects the longest matching location prefix
  - regardless of order
  - before checking regex locations
  - modifier may keep regex locations from being checked
- NGINX checks regex locations in order listed
  - selects first matching
  - modifier determines case-sensitivity

Knowledge Check

Which location will NGINX select to serve the following request? http://example.com/app1/logo.png

- ☒ `location = / { }`
- ☑ `location ~* \.(png|jpg)$ { }`
- ☒ `location /app1 { }`

Knowledge Check

Which location will NGINX select to serve the following request? http://example.com/app1/images/logo.png

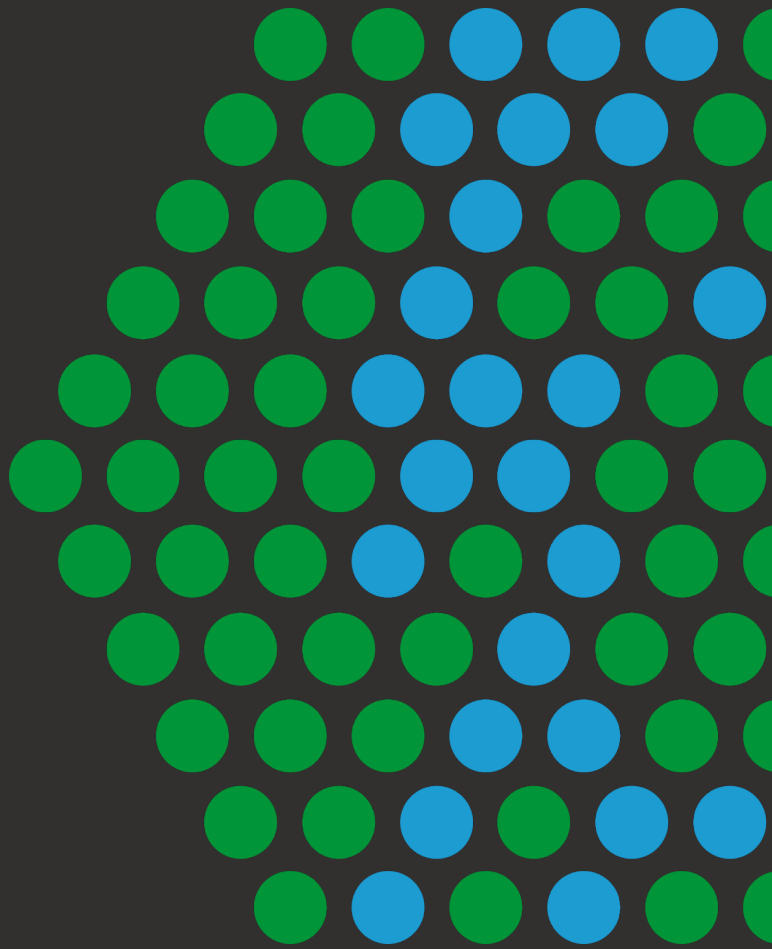
- ☒ `location = / { }`
- ☒ `location = /app1/ { }`
- ☑ `location /app1/images { }`
- ☒ `location = /app1/logo.png { }`

Lab 5 – Location Blocks

NGINX
Part of F5

Module 6

Using Variables



Module 6 Objectives

- Identify NGINX's predefined variables
- Understand Variable Scope
- Demonstrate variable usage during request header processing
- Demonstrate custom variable assignment

NGINX Variable Syntax

Syntax: `$<variable_name>`

Examples:

Variable Name	Meaning
$remote_addr	Client address
$request	Original request
$status	Response status
$uri	URI in request

http://nginx.org/en/docs/varindex.html

Example Use Cases

1. Conditional Proxying
2. Conditional Logging
3. Mapping HTTP Headers
4. Gathering and Storing Metrics
5. Selective Authorization
6. Dynamic Blacklisting
7. A/B Testing
8. Tracing request IDs

Core Module Variables

Variable Name	Use
$host	Host name from HTTP header
$request_uri	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
$args	Arguments in the request line

Special Variables

- $arg_<name>
- $cookie_<name>
- $http_<name>

Examples

```
$arg_sessionid
$cookie_mycookie
$http_500
```

Request Variables Example

Request: http://localhost:8080/test?arg=1

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

Variable Usage Examples:

- Force Traffic to HTTPS:

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

- Proxy an HTTPS request with original request URI values:

```
server {
    listen 443 ssl;
    proxy_pass http://example.com:8080$request_uri;
}
```

Useful Troubleshooting Variables

Variable	Definition
$upstream_connect_time	Connection to upstream/backend
$upstream_header_time	First byte response header
$upstream_response_time	Last byte response body
$request_time	Total time of request

Note: These variables are not logged by default. Add them to your log_format directive to view them.

Create Custom Variables

Syntax: `set $<variable> <value>`

Allows for custom variable definition and assignment

Use cases:

- Create conditional variable for allowing/denying requests
- Create request variable for routing to specific backend

Variable Assignment

- Occurs at **runtime**
- Runs in own request container
- Can be
  - strings
  - numbers (integer/float)
  - other variables

```
set $variable_name value;

set $foo "hello world";
set $bar 101;
set $combo $foo
```

Variable Scope

- Variables can exist in any context
- Scope refers to the placement in the config context

```
server {
        listen 80;
        location /1 {
                set $foo "hello world";
        }
        location /2 {
                set $bar "hello darkness my old friend";
        }
}
```

Module Summary

- NGINX has pre-defined variables, special variables, and debugging variables
- The set directive allows for custom variable definition and assignment
- Variable assignment occurs at runtime
- Variable scope depends on context placement

Knowledge Check

```
server {
        listen 8080;
        server_name server1;
        location /test1 {
                return 200 "$foo";
        }
        location /test2 {
                set $foo hello;
                return 200 "$foo";
        }
}
server {
        listen 8081;
        set $foo 42;
        server_name server2;

        location /test3 {
                return 200 "$bar";
        }
}
```

Which prefix location returns a value for $foo

- ☒ `location /test1`
- ☑ `location /test2`
- ☒ `location /test3`

Knowledge Check

```
server {
        listen 80;
        server_name server1;
        set $foo 54;

        location / {
                set $bar 31;
                return 200 "$foo";
        }
        location /test1 {
                set $foo 456;
                return 200 "$foo";
        }
        location /test2 {
                set $foo $bar;
                return 200 "$foo";
        }
}
```

What is the value of $foo if the client request is: http://server1/test1

- ☐ 31
- ☑ 456
- ☐ null
- ☐ 54

Knowledge Check

```
server {
        listen 80;
        server_name server1;
        set $foo 54;

        location / {
                set $bar 31;
                return 200 "$foo";
        }
        location /test1 {
                set $foo 456;
                return 200 "$foo";
        }
        location /test2 {
                set $foo $bar;
                return 200 "$foo";
        }
}
```

What is the value of $foo if the client request is: http://server1/test2

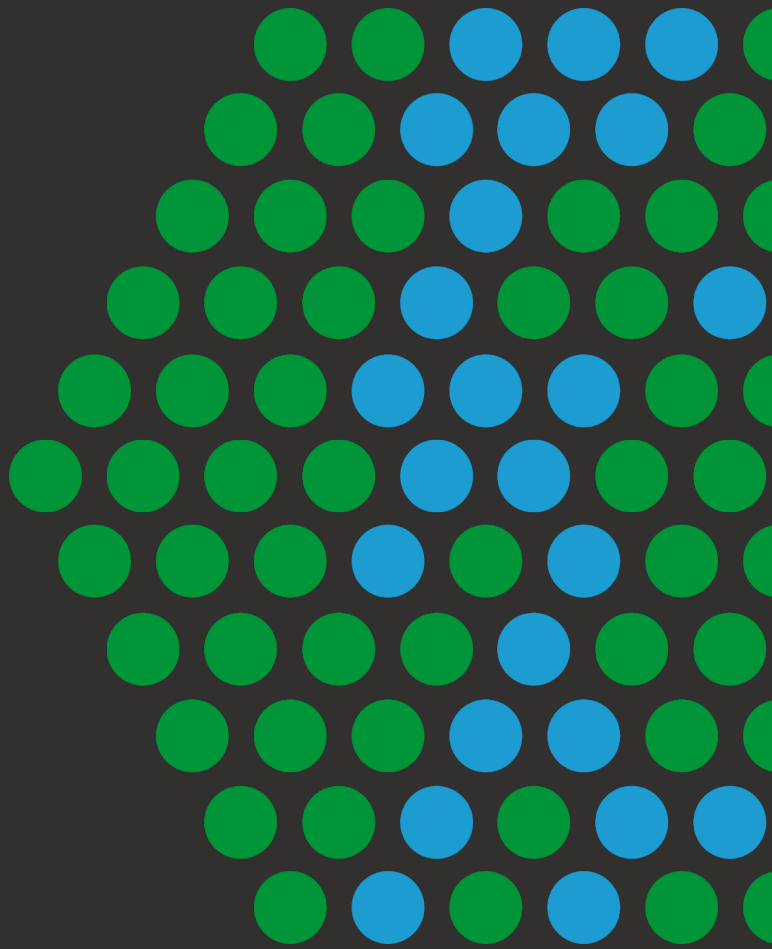
- ☐ 31
- ☐ 456
- ☑ null
- ☐ 54

Lab 6 – Using Variables

NGINX
Part of F5

Module 7

Logging



Module 7 Objectives

- Set up logging to audit connections to NGINX
- Demonstrate use cases for log levels
- Configure error and access logs
- Use predefined variables in the log_format directive to log specific connection metrics

Error & Access Logs

- **error_log** – messages from NGINX
  - detail level managed by `level` parameter
- **access_log** – audit incoming connection details
  - detail level managed by `log_format` directive

error_log Directive

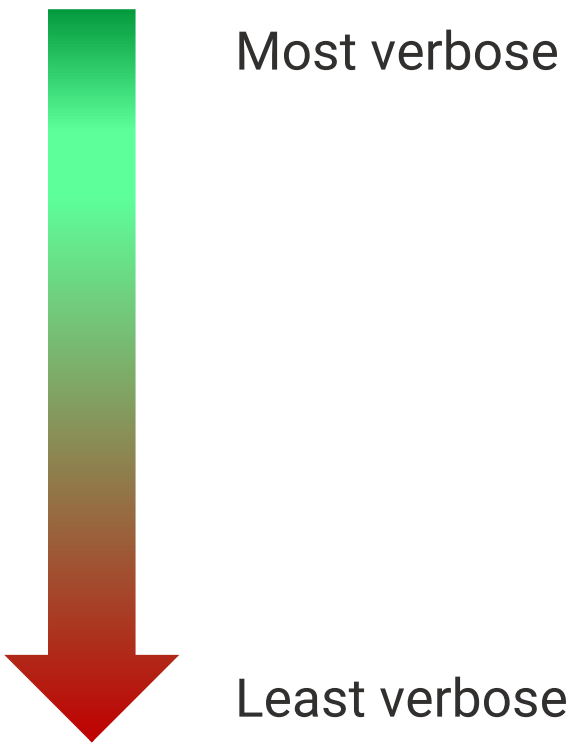
- Configures error log settings
- Syntax: **`error_log <file>`** `[level]`
- Default location: /var/log/nginx

Error Log Levels

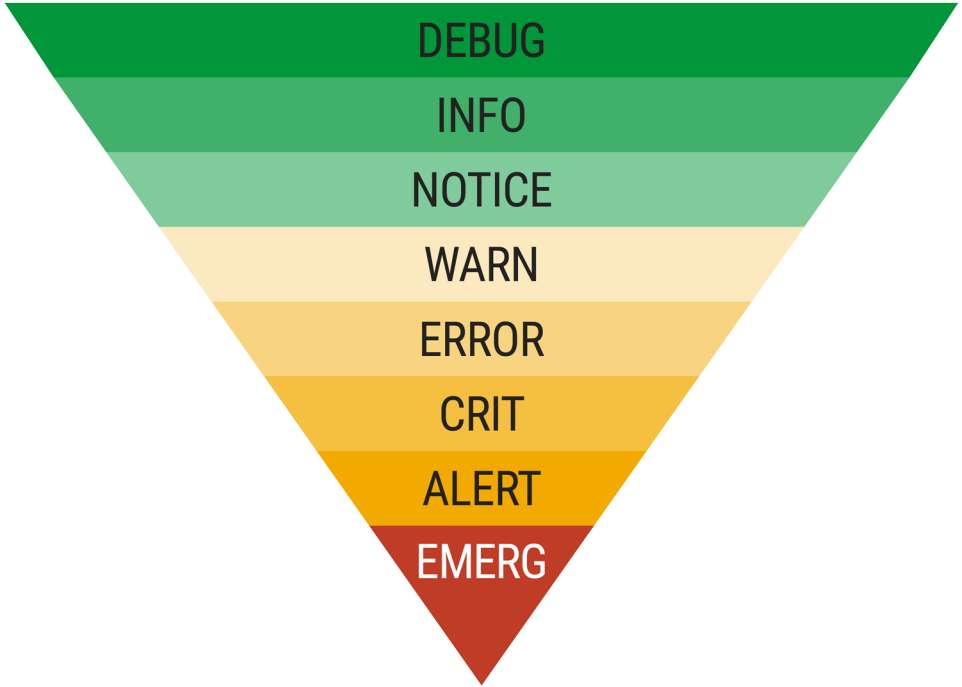
Log Level	Meaning
`debug`	Detailed trace, almost everything logged
`info`	General information
`notice*`	Something normal
`warn`	Something strange
`error**`	Unsuccessful
`crit`	Important issue
`alert`	Fix now
`emerg`	Only events that render system useless

*specified in /etc/nginx/nginx.conf but **if no level specified defaults to error

Error Log Levels

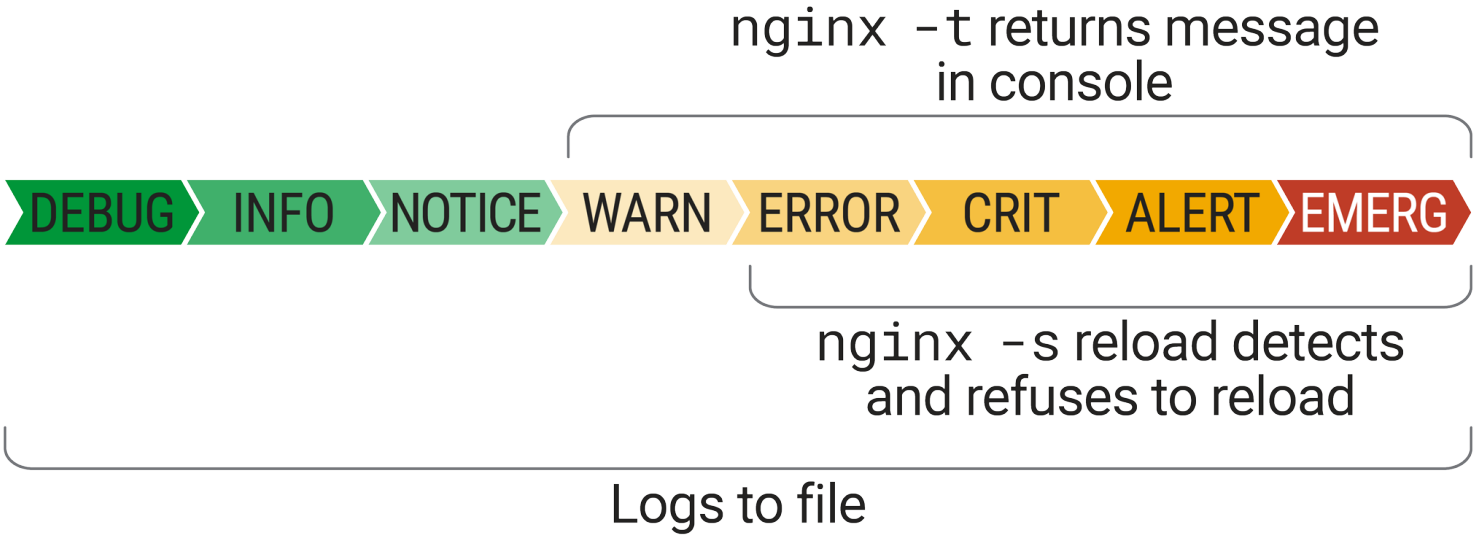


Most verbose



Least verbose

Output Results



Error Log Example

- Must provide absolute file path

```
server {
     listen 80;
     server_name server1.com;
     error_log /var/log/nginx/server1_info.log info;

     location /error {
          error_log /var/log/nginx/server1_error.log error;
     }
}
```

Access Log Directive

- Syntax: `access_log <file> <type/name>`
- Logs all attempts to access server(s)
- Can determine connection details
  - Connection time
  - Request/Response time
  - Cache-control headers
- Default log location
  - /var/log/nginx/access.log

Access Log Format Directive

- Syntax: `log_format <name> <variables>`
- Defines connection details
  - NGINX pre-defined variables
  - Custom variables
  - Examples
    - Client IP address
    - Response status
    - Time stamps

Combined Log Format

Variable	Meaning
$remote_addr	Client address
$user	Username supplied with basic authentication
$time	Local time
$request_URI	Full request URI
$response	Response status
$bytes_sent, $bytes_received	Bytes sent/received to/from client excluding response header
$http_user_agent, $http_referrer	App type, OS, other connection details

Access Log Example

```
log_format custom “Request: “$request_uri\n Status: $status\n
Backend-Address: $upstream_addr”;

server {
     listen 80;
     server_name server1.com;
     access_log /var/log/nginx/access.log combined;

     location /private {
          access_log /var/log/nginx/private_access.log custom;
     }
}
```

JSON Logging

```
log_format custom_json escape=json '{'
        '"time": "$time_local",'
        '"high_level": {'
                '"remote_addr": "$remote_addr",'
                '"remote_user": "$remote_user",'
                '"request": "$request",'
                '"http": {'
                '"http_referer": "$http_referer",'
                '"http_user_agent": "$http_user_agent",'
                '"http_x_forwarded_for": "$http_x_forwarded_for"' '
                }'
        '}'
'}';
```

Log Delimiter

Delimiter	Meaning
,	Request processed by multiple servers
;	Internal redirect between upstreams
0	Unable to reach upstream
-	Internal error or cached value

Example Access Log

```
GET /home HTTP/1.0 200 host3/home 173.186.135.227, 172.31.30.59
EXPIRED


cache:  HIT

upstream_addr: -

request:  GET / HTTP/1.1
```

Customize your Access Log: upstream_time

Log format for upstream time:

```
log_format upstream_time ‘”$request_time”
                         “$upstream_connect_time”
                         ”$upstream_header_time”
                         “$upstream_response_time”’;
```

Logging Best Practices

```
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

```
#!/bin/bash

#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)
```

Syslog Protocol

Use syslog protocol log messages to syslog servers

```
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,
severity=info combined;
```

Module Summary

- Error logs are governed by the level of error
- Access logs are governed by the log_format directive
- Best practices
  - Set up individual log files per server
  - Rotate logs
- Use syslog protocol to send logs to syslog servers

Knowledge Check

Access logs require (choose one)

- ☑ A log_format command
- ☒ A “combined” type
- ☒ JSON formatting
- ☒ At least “alert” level logging

Knowledge Check

When you use the “**error**” log level on an **error_log** directive, you log what level of errors?

- ☒ warn, crit, error, alert
- ☒ debug, info notice, warn, error
- ☑ emerg, alert, crit, error
- ☒ warn, alert, crit

Knowledge Check

In the following log line example, what does the **comma** indicate?

```
GET /home HTTP/1.0 200 host3/home 173.186.135.227,
172.31.30.59 EXPIRED
```

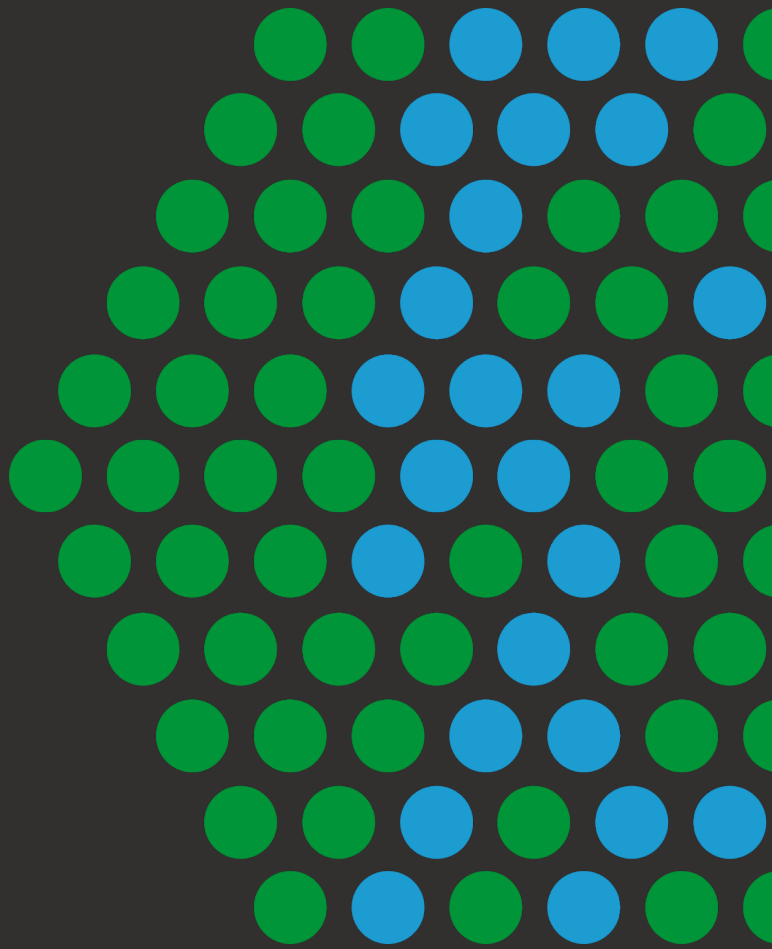
- ☐ The variable is the upstream address and because the reverse proxy cache served the response, the server is confused.
- ☑ The request went through multiple servers.
- ☐ There was an internal redirect through upstream servers.
- ☐ There was an internal error

Lab 7 - Logging

NGINX
Part of F5

Module 8

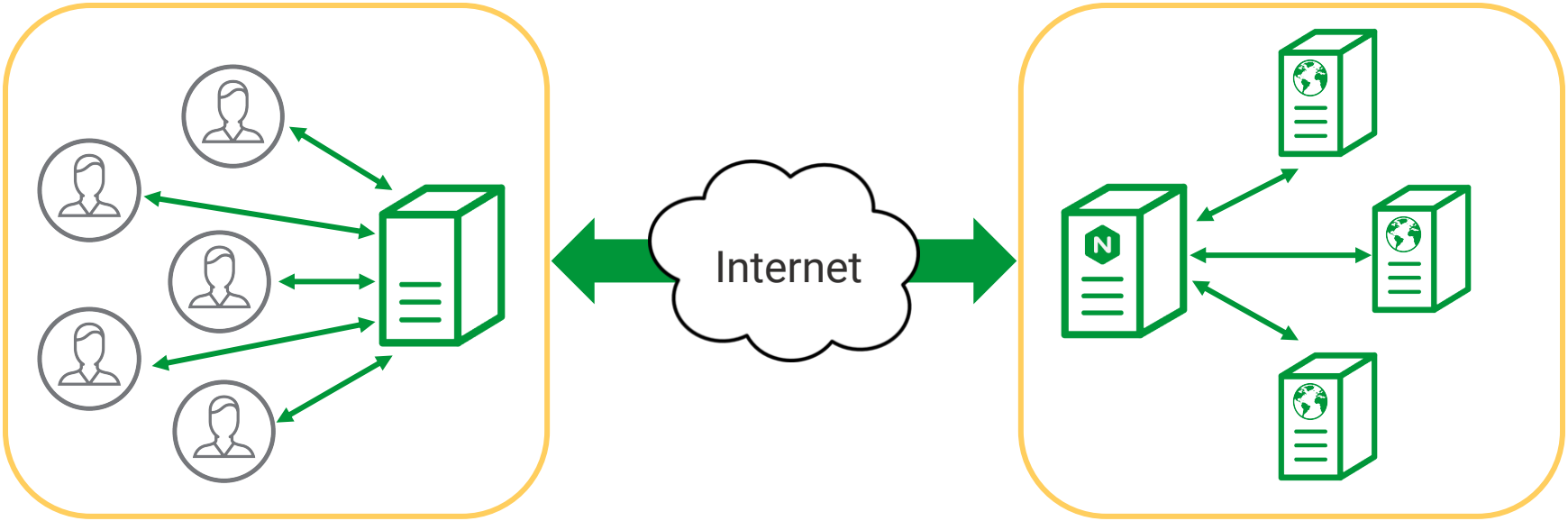
Proxying Requests



Module 8 Objectives

- Configure NGINX as a reverse proxy
- Identify HTTP request/response behavior while proxying requests

Reverse Proxy Server



Forward Proxy

Reverse Proxy

proxy_pass Directive

- Syntax: `proxy_pass <destination>;`
- Used in server or location contexts
- Example request: http://www.example.com/

```
location / {
    proxy_pass http://192.4.1.4/application1;
}
```

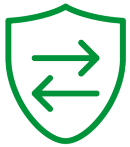
Proxying non-HTTP Requests

Directives for translating HTTP to other request protocols

Directive	Passes to:
fastcgi_pass	FastCGI server
uwsgi_pass	uswgi server
scgi_pass	SCGI server
grpc_pass	gRPC server
memcached_pass	memcached server

Note: See NGINX reference docs for further info.

Proxy with a Path - Replace

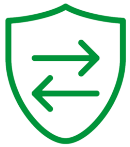


Request: http://www.example.com/application1/usr1/index.php

```
server {
  listen 80 default_server;
  server_name www.example.com;
  root /usr/share/nginx/html;

  location /application1 {
     proxy_pass http://192.168.1.2:8080/otherapp
     }
}
```

Proxied request:
http://192.168.1.2:8080/otherapp/usr1/index.php

Proxy Without a Path - Append

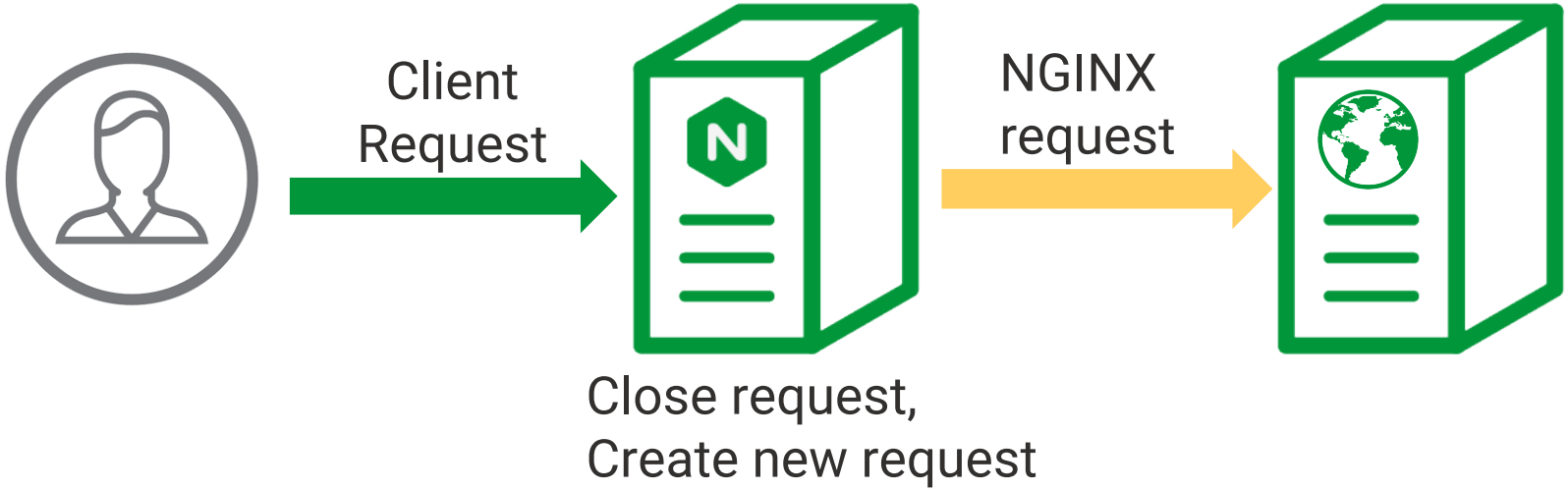


Request: http://www.example.com/application2/usr2/index.php

```
server {
     listen 80 default_server;
     server_name www.example.com;
     root /usr/share/nginx/html;

     location /application2 {
          proxy_pass http://192.168.1.2:8080;
     }
}
```

Proxied request:
http://192.168.1.2:8080/application2/usr2/index.php

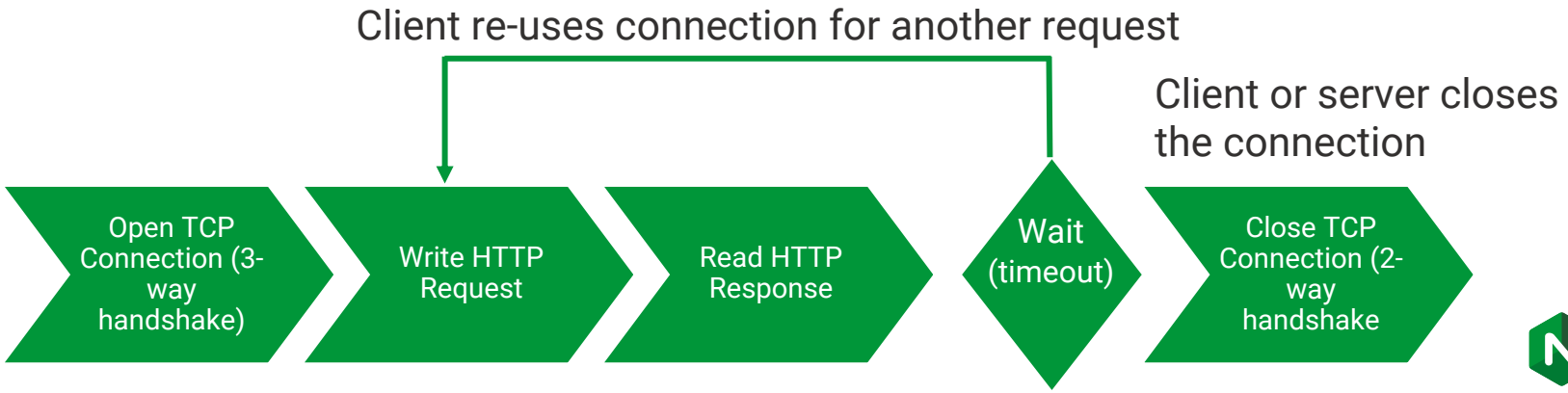
NGINX Default Proxy Behavior



Keepalive Process

Normal TCP Handshake:

Keepalive Connection:



Client Keepalive Directives

```
http {
    keepalive_requests 150;
    keepalive_timeout 30s;
...
}
```

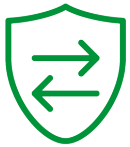
Proxying or Redefining Request Headers

Syntax: **proxy_set_header** <HTTP request header field name> <new value>

```
server {
  listen 80;
  proxy_set_header Host $host;
  proxy_set_header X-Real-IP $remote_addr;
  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    location / {
      proxy_pass http://backend-server.com;
}
```

https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

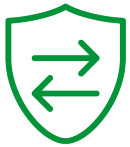
add_header Directive



Syntax: `add_header <HTTP response header field name> <new value>`

```
server {
    listen 80;
    add_header X-Frame-Options DENY;
}
```

try_files Directive



- Searches context for a directory or file and serves response from disk

Syntax: `try_files <variable(s)>`

Example:

```
location /test {
    try_files $uri $uri/ $uri.html 404.html;
}
```

Module Summary

- Use the proxy_pass directive to use NGINX as a reverse proxy
- NGINX can proxy non-HTTP requests
- Keepalive directive keeps an initial client TCP Connection open to send multiple requests through one connection
- Redefine headers with proxy_set_header directive as needed
- Add headers for client response as needed

Knowledge Check

The proxy_pass directive may be used in the HTTP or server context.

- ☒ True
- ☑ False

Knowledge Check

With the following request and config, what is the proxied request?
http://www.example.com/app1/projects/p1.html

```
server {
    listen 80 default_server;
    server_name www.example.com;
    root /usr/share/nginx/html;
        location /app1 {
            proxy_pass http://MyServer;}
}
```

- ☐ http://MyServer/projects/p1.html
- ☑ http://MyServer/app1/projects/p1.html
- ☐ http://MyServer/usr/share/nginx/html/projects/p1.html
- ☐ http://www.example.com/projects/p1.html

Knowledge Check

With the following request and config, what is the proxied request?
http://www.example.com/app1/projects/p1.html

```
server {
    listen 80 default_server;
    server_name www.example.com;
    root /usr/share/nginx/html;
        location /app1 {
            proxy_pass http://MyServer/app2;}
}
```

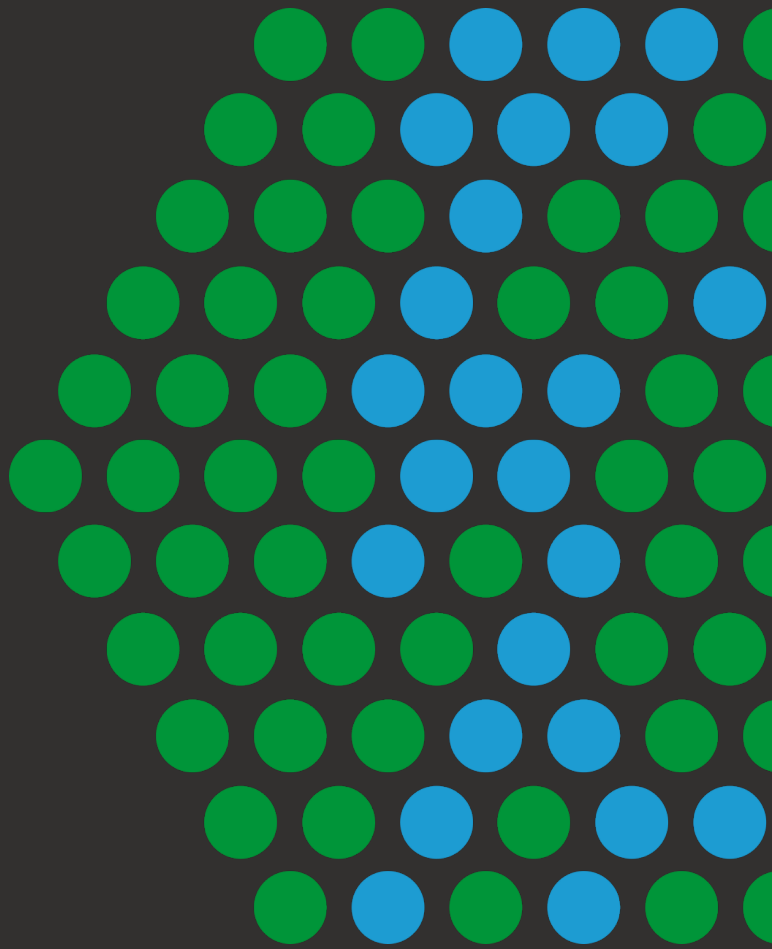
- ✅ http://MyServer/app2/projects/p1.html
- ❌ http://MyServer/app1/app2/projects/p1.html
- ❌ http://MyServer/usr/share/nginx/html/projects/p1.html
- ❌ http://www.example.com/app2/projects/p1.html

Lab 8 – Proxying HTTP Requests

NGINX
Part of F5

Module 9

Routing HTTP Requests



Module 9 Objectives

- Use directives to reroute traffic
- Configure URL rewrites
- Describe how NGINX processes rewrite requests

Use Cases for Routing Traffic

- Domain name changes
- Relocated resources
- Manage processing flow

Methods of Routing Traffic

Directive:

- **`alias`**
- **`return`**
- **`rewrite`**

The alias Directive

- Syntax: **`alias`** ***`path`***
- Specifies a replacement path

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

  location /images {
    alias /data/media/pictures;
  }
  location /videos (
    alias /data/media/mp4;
  }
}
```

Regex with alias Directive

- Define capture group using Perl regular expression
- Use the capture group in the `alias` directive

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

The return Directive

Syntax: **return *code url*;**

return *url*;

- Returns an HTTP response code and/or URL, and string

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

    location /test {
        return http://test-server$uri;
    }
}
```

The rewrite Directive

- Syntax: `rewrite regex replacement [flag];`
- Regular expression matched against URI
- Replacement string rewrites the URI
- Flags
  - `last` – stop processing, search for changed URI (usually server context)
  - `break` — stop processing (usually location context)
  - `redirect` — returns temporary redirect with 302
  - `permanent` — returns permanent redirect with 301

The rewrite Process

- Executed sequentially
- Executed upon server selection
- Rewrites in higher contexts are executed first
- Flags can stop processing
- Blog

https://www.nginx.com/blog/creating-nginx-rewrite-rules/

Rewrite Example

- Request: http://server/users/james
- Server rewrites as SQL query: /show?user=james
- If no matching username, 404 code returned

```
location /users/ {
     rewrite ^/users/(.*)$ /show?user=$1 last;
     return 404 /40x.html;
}
```

last vs. break

Rewrite only uri, it doesn't work on ip addr

```
server {
     rewrite ^(/dwnld/.*)/media/(.*)\..*$  $1/mp3/$2.mp3 last;
     rewrite ^(/dwnld/.*)/audio/(.*)\..*$  $1/mp3/$2.ra last;
     return 403;
}
```

```
server {
...
  location /dwnld/ {
     rewrite ^(/dwnld/.*)/media/(.*)\..*$  $1/mp3/$2.mp3 break;
     rewrite ^(/dwnld/.*)/audio/(.*)\..*$  $1/mp3/$2.ra break;
     return 403;
     }
}
```

Return vs. Rewrite

Returns*

- Simpler
- Uses variables
- Halts processing
- Less processing overhead

❖ Use when

- ➢ Can build with `$variables`
- ➢ New URL is good for every request that matches

Rewrites

- More complex
- Uses regex
- Halts processing with flags
- More processing overhead

❖ Use when

- ➢ No corresponding NGINX `$variables`
- ➢ Need to change or add elements in the path

*Preferred method

Module Summary

- Use cases for rerouting traffic include changes to:
  - Domain names
  - Resources
  - Request processing flow
- Use the **alias** directive to set a replacement path for content
- Use the **return** directive to reroute a request to a different server or location
- Use the **rewrite** directive to reroute traffic using regular expressions

Knowledge Check

With the following request and config, where is it served from?
http://www.example.com/images/cat.jpg

```
server {
    listen 80 default_server;
    root /usr/share/nginx/html;
        location /images {
            alias /data/media/pictures;}
}
```

- ☐ /images/data/media/pictures/cat.jpg
- ☑ /data/media/pictures/cat.jpg
- ☐ /usr/share/nginx/html/data/media/pictures/cat.jpg
- ☐ /home/public_html/data/media/pictures/cat.jpg

Knowledge Check

In general practice, which rewrite flag is best to stop rewrite processing in the server context and move on to the location context?

- ☒ break
- ☒ redirect
- ☑ last
- ☒ permanent

Knowledge Check

Which of the following directives is best to use when you have to make conditional changes to a request?

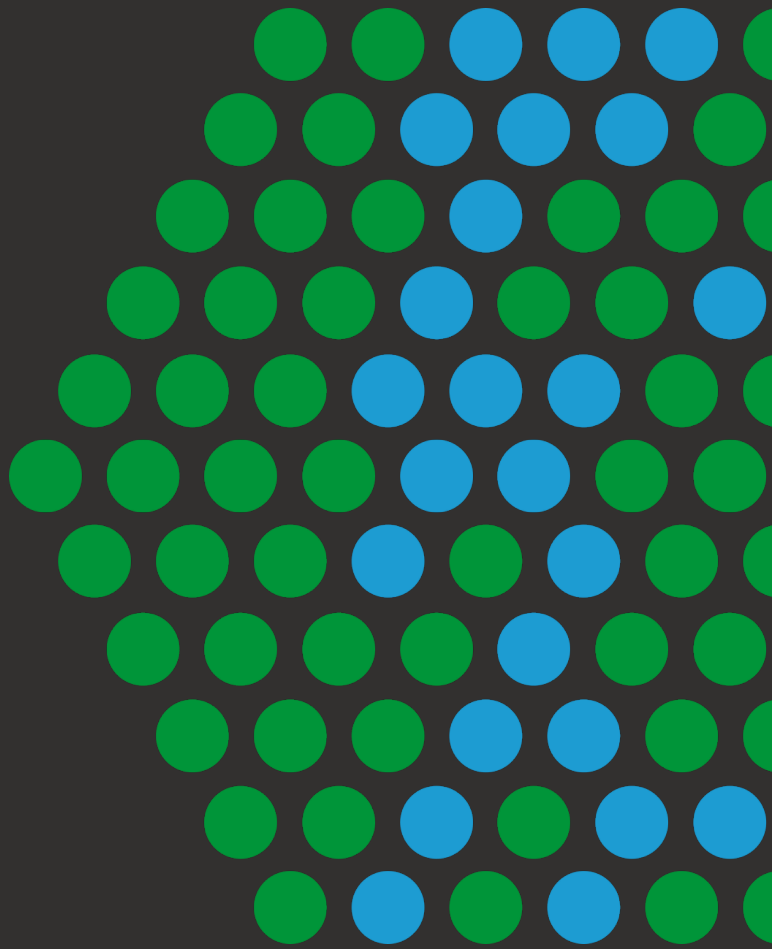
- ☒ alias
- ☒ return
- ☑ rewrite
- ☒ sub_filter

Lab 9 – Routing HTTP Requests

NGINX
Part of F5

Module 10

Mapping Variables



Module 10 Objectives

- Identify use cases for the map directive
- Demonstrate usage of the map directive
- Identify use cases for the if directive
- Demonstrate usage of the if directive

The map Directive

- Syntax: **`map $var1 $var2`** *`{value value}`*
- Maps value of one variable to another

```
map $uri $path {
    /test1 /path1;
    /test2 /path2;
    /test3 /path3;
}
server {
    listen 80;
    location /test2 {
      return 200 “$path \n”;
    }
}
```

Default Value Variable

- Default values protect from translation errors

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

Regex in Maps

- The map directive supports both relative and positional captures

```
map $uri $path {
    ~*/test/.*\.php$ @php;
    default /;
}
```

Use Case: Conditional Logging

- **`$loggable`** (custom) variable determines whether to log an event

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

server {
   listen 80;
   access_log /path/to/access.log combined if=$loggable;
...
}
```

The if Directive

- Syntax: `if (condition) { <content_handler>; }`
- Server or location context
- Should be used with care
- Use a map if possible
- Safe use cases for location:
  - `return`
  - `rewrite`

https://www.nginx.com/resources/wiki/start/topics/depth/ifisevil/

if Directive Example

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

        location / {
                if ($request_method = POST ) {
                return 405;
                }
                proxy_pass http://localhost:8080;
        }
}
```

Module Summary

- The map directive allows you to set up a conditional relationship between a list of variables
- The map directive may be used with
  - A default value
  - Regular expressions
  - A custom `$loggable` variable
  - The `if` directive
- The if is a directive with a conditional in parenthesis

Knowledge Check

With the following request and config, what is the value of **$path**?
https://www.example.com/test5

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

- ☑ /path
- ☒ /path1
- ☒ /path2
- ☒ /path3

Knowledge Check

The **`if`** directive, while useful, can be problematic. In which uses below is it safe to use?

- ☒ `add_header`
- ☑ `return`
- ☒ `alias`
- ☑ `rewrite`

Knowledge Check

A regular expression can set the value of a map variable.

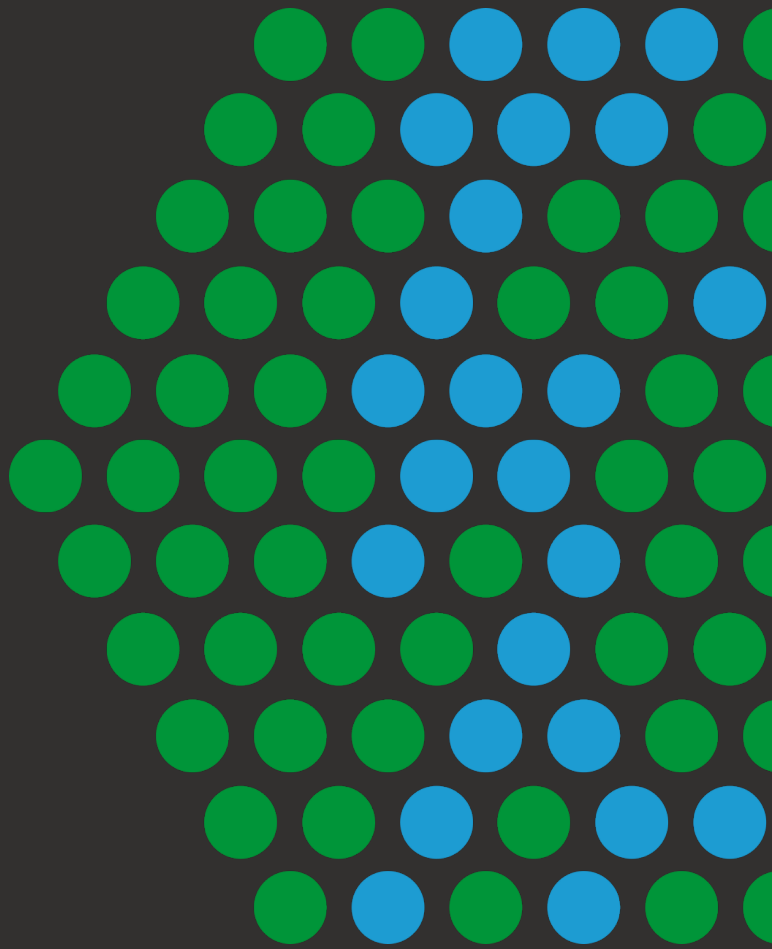
- ☑ True
- ☒ False

Lab 10 – Mapping Variables

NGINX
Part of F5

Module 11

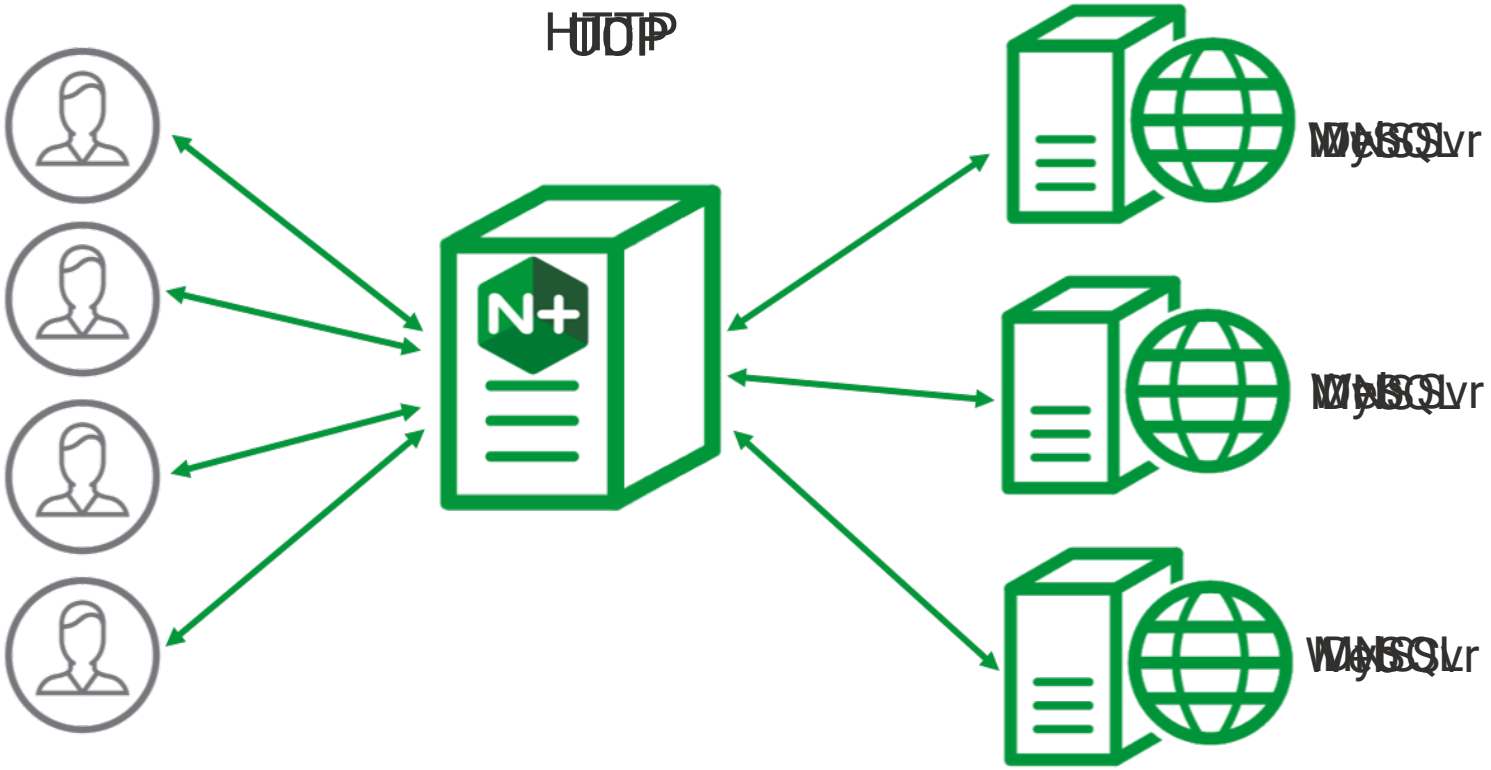
Load Balancing



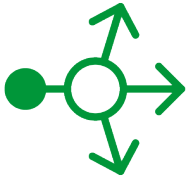
Module 11 Objectives

- Identify load balancing methods
- Set up and test load balancing

Load Balancing Use Cases

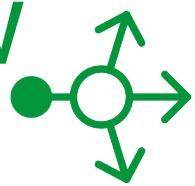


Load Balancing Methods



- `round robin`
- `least_conn`
- `least_time` N+
- `hash`
- `ip_hash`
- `random` N / N+

HTTP Load Balancing Overview



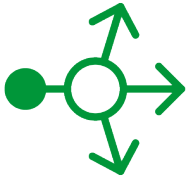
- Define an **`upstream`** - group / pool of servers
- Use **`proxy_pass`** directive
- Servers can be defined by:
  - UNIX socket
  - Domain name
  - IP:Port

HTTP Configuration Example

- `upstream` directive lists servers in pool
- `proxy_pass` directive forwards requests to upstream server pool

```
upstream myServers {
    server training1.example.com;
    server training2.example.com:8080;
}
server {
    listen 80 default_server;
     location / {
        proxy_pass http://myServers;
    }
}
```

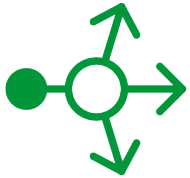
Server Weights



Weight is used to distribute load among backend servers

```
upstream myServers {                    Load
    server backend1 weight=5;           5/12
    server backend2 weight=3;           3/12
    server backend3 weight=4;           4/12
}                                 12
```

Server Fails and Timeouts



- **`max_fails`** - failed connection attempts before a server is marked unavailable
- **`fail_timeout`**
  - Time limit for ~~consecutive~~ failures
  - Duration for which server is unavailable

```
upstream myServers{{
  server backend1 weight=5 max_fails=10 fail_timeout=90s;
  server backend2 weight=3 max_fails=4 fail_timeout=60s;
  server backend3 weight=4 max_fails=2 fail_timeout=30s;
}
```

max_conns /queue

```
upstream backend {
    server backend1.com max_conns=300;
    server backend2.example.com;
    queue 100 timeout=90;
}
```

hash Method Example

```
upstream myServers {
    hash $request_uri consistent;

        server backend1.com;
        server backend2.com;
        server backend3.com;

}
```

Add or remove server will caused hash remapping

consistent: Maintain the same hash value (not 100%)

ip_hash Method Example

```
upstream myServers {
    ip_hash;    based on the client ip addr

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

Note: This method only uses the first 3 octets of the IPv4 address.

least_conn Method Example

```
upstream myServers {
    least_conn;

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

least_time Method Example

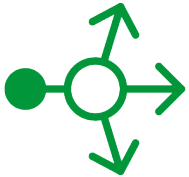
```
upstream myServers {
    least_time header;
    least_time last_byte;
    least_time last_byte inflight;

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

random Method Example

```
upstream backend {
    random;two;least_time=last_byte;
        server backend1.com;
        server backend2.com;
        server backend3.com;
        server backend4.com;
}
```

See: https://www.nginx.com/blog/nginx-power-of-two-choices-load-balancing-algorithm/

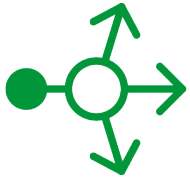
Load Balancing Comparison



Layer 4	Layer 7
`stream` context	`http` context
UDP and TCP protocols	HTTP protocol
No header inspection so no `location` context	Header inspection allows `location` context (uri match)
Health checks based on send / expect parameters	Health checks based on response header / body

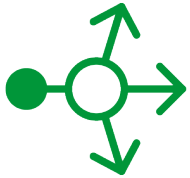
Link to Contexts

TCP Load Balancing Example



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

UDP Load Balancing Example



```
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_connect_timeout 2s;
            proxy_responses 1;
            error_log logs/dns.log warn;
    }
}
```

Module Summary

- NGINX can load balance layer 4 and 7 traffic
- Use the upstream and proxy_pass directives
- Use the stream context for UDP and TCP load balancing
- Choose from 5 load balancing methods
  - Round Robin
  - Hash
  - IP Hash
  - Least Connections
  - Least Time
  - Random

Knowledge Check

Which load balancing method is the configuration below using?

```
upstream myServers{
    server training.example.com;
    server.training.example1.com:8080;
}
server {
    listen 80 default_server;
    location / {
        proxy_pass http://myServers;
    }
}
```

- ☒ IP hash
- ☒ Weighted hash
- ☒ Random
- ☑ Round robin

Knowledge Check

Which of the following are parameters of the `least_time` load balancing method?

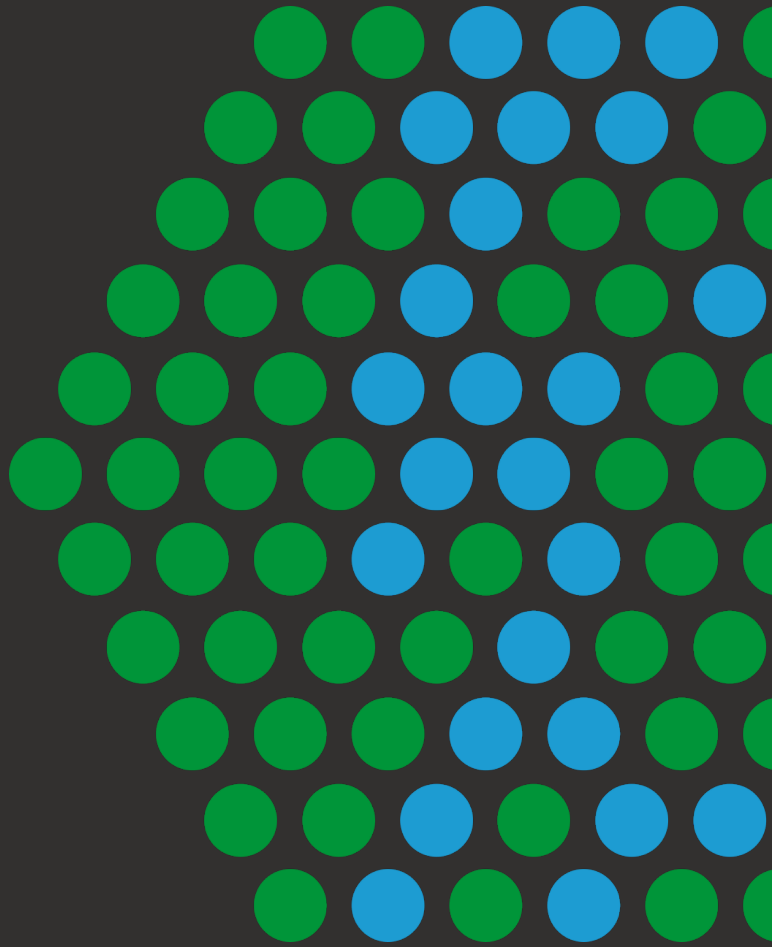
- [x] `header`
- [x] `last_byte`
- [ ] `first_byte`
- [x] `inflight`

Lab 11 – Load Balancing

NGINX
Part of F5

Module 12

Using the NGINX API



Module 12 Objectives

- “Live Activity Monitoring” – Configure NGINX Plus to view performance metrics – NGINX Plus dashboard
- Use the NGINX Plus API to configure back end servers

NGINX Plus Dashboard Setup

- `api` directive
- `zone` directive (upstream context)
- `status_zone` directive (server context)
- `dashboard` location

NGINX API Documentation

NGINX Plus REST API 3.0

[Base URL: /api/3]
/swagger-ui/nginx_api.yaml

NGINX Plus REST API provides access to NGINX Plus status information, on-the-fly configuration of upstream servers and key-value pairs management for http and stream.

Schemes
HTTP

General Info

GET / Return list of root endpoints

GET /nginx Return status of nginx running instance

Processes

GET /processes Return nginx processes status

DELETE /processes Reset nginx processes statistics

Swagger UI: https://demo.nginx.com/swagger-ui/

NGINX api Directive

- write
- allow/deny
- state

```
server {
    listen 8080;
    location /api {
        api;write=on;
        allow 192.168.0.0/16;
        deny all;
    }
}
```

state Directive

- Specifies a file that keeps the state of the upstream group

```
upstream myServers {
    zone backend 64k;
    server backend1 weight=1;
    server backend2 weight=3;
    state /etc/nginx/conf.d/backend.state;
}
```

zone Directive

- Shares counters for key values in upstream context

```
upstream myServers {
    zone backend 64k;
    server backend1 weight=1;
    server backend2 weight=3;
    state /etc/nginx/conf.d/backend.state;
}
```

status_zone Directive

- Shares counters for key values in server context
- R19 shares stats also per location context

```
server {
    listen 8081;
    root /server/backend1;
    status_zone us;
}
server {
    listen 8082;
    root /server/backend2;
    status_zone apac;
}
```

Dashboard Location

```
server {
    listen 8080;
    location /api {
        api;
    }
    location /dashboard.html {
        root /usr/share/nginx/html;
}
```

Using the NGINX API

- View, modify, remove upstream servers at runtime
- No reload to effect changes
- Use with the state directive for permanence
- Don’t have to bring servers down to test a config

API Requirements

- **`/api`** location with write access
- **`zone`** directive
- Use **`state`** directive to save configs*

```
upstream myServers {
    zone backend 64k;
    state /etc/nginx/backend.state
    server localhost:8081;
    server localhost:8082;
}
server { ...
    location / {
        proxy_pass http://myServers;
    }
    location /api {
        api write=on;
        allow some-ip-range;
        deny all;
     }
}
```

*optional

Common API Endpoints

Endpoint	Returns
/nginx/	Build info
/http/	HTTP info
/http/upstreams/	Upstream info
/http/upstreams/<name>/servers/	Server info
/http/upstreams/<name>/servers/<id>	Individual server info

http://nginx.org/en/docs/http/ngx_http_api_module.html

Example Curl Command

- curl -X GET "http://localhost/api/5/http/upstreams/myServers/servers/" -H "accept: application/json" | jq

https://demo.nginx.com/swagger-ui/

Example Curl Response

- Returns the server config
- Each server in an upstream server group


```
[
  {
    "id": 0,
    "server": "10.0.0.1:8088",
    "weight": 1,
    "max_conns": 0,
    "max_fails": 0,
    "fail_timeout": "10s",
    "slow_start": "10s",
    "route": "",
    "backup": false,
    "down": false
  }
]
```

Module Summary

- NGINX Plus provides a real-time dashboard for server pool metrics
- Allows access to configure upstreams via its API
  - Configure and test without reloads
  - Save state

Knowledge Check

Which of the following are required to set up monitoring using The NGINX Plus dashboard?

- ☒ api parameter
- ☑ zone directive
- ☑ dashboard location
- ☑ status_zone directive
- ☑ api directive
- ☒ write=on parameter

Knowledge Check

Which configuration(s) can help make the api directive secure?

- ☒ `Zone backend 64k;`
- ☑ `allow 127.0.0.1;`
  `deny all;`
- ☒ `state /etc/nginx/conf.d/backend.state`
- ☑ `api write=off;`

Knowledge Check

True or False? The NGINX API returns information in Json format.

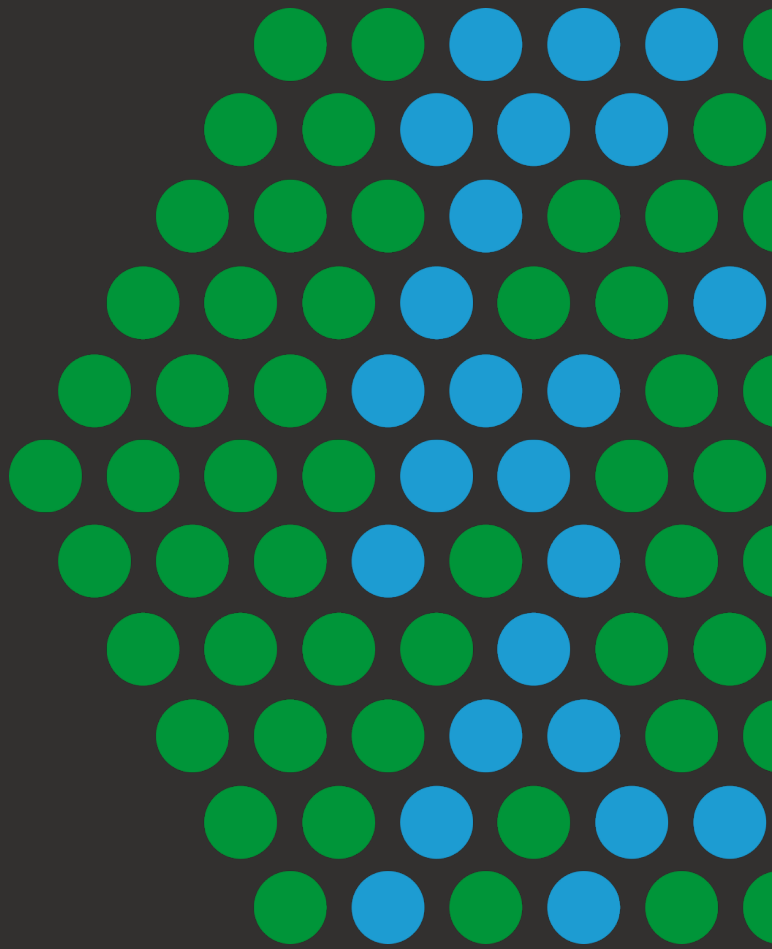
- ☑ True
- ☒ False

Lab 12 – Using the NGINX API

NGINX
Part of F5

Module 13

Health Checks



Module 13 Objectives

- Set passive health checks
- Set active health checks based on match block conditions

Passive Health Check

- Parameters on `server` directive
  - `fail_timeout`
  - `max_fails`

```
upstream myServers {
    server backend1;
    server backend2 max_fails=3 fail_timeout=60s;
}
```

Active Health Checks

- Request sent by NGINX to check upstream group
- Resides in **`location`** context
- Under **`proxy_pass`** directive
- Verifies a server is healthy
- Can be used with parameters and a list of conditions
- **zone** directive - upstreams shared memory area

health_check syntax

```
upstream myUpstreams {
    zone backend 64k;
    server backend1 weight 1;
    server backend2 weight 2;
}
server {
    listen 8080;
    location / {
        proxy_pass http://myUpstreams;
        health_check [parameters];
```

health_check Parameters

Parameter	Use
`interval`	Time between health checks
`fails`	Max number of fails
`passes`	Min number of passes
`uri`	Path to a test file

match Directive / match Block

- Defines conditions for a health check
- Conditions can be based on
  - Response codes (status codes)
  - Header values
  - Text of documents
- You may use multiple conditions

Match Block Syntax

```
match server_ok {
    status <value>;
    header <value>;
    body <value>;
    require $variable …;
}
server { ...
    location / {
        proxy_pass http://backend;
        health_check match=server_ok;
        }
    }
}
```

match Block Test Options

- **`status`** — checks response code of server
  - `status 200;`
  - `status ! 400 402;`
  - `status 200-399;`
- **`header`**
  - `header Content-Type = text/html;`
  - `header Cache-Control;`
  - `header Connection ~ close;`

match Block Test Options

- `body`
  - `body ~ “Hello World”`
  - `body !~ “Hello World”`
- `require $variable ...;`
  - Call a variable using NGIXN javaScript

Match Block Example

```
match conditions {
    status 200;
    header Content-Type = text/html;
    body !~ “maintenance”;
}

server {
    location /{
    proxy_pass http://myServers;
    health_check match=conditions fails=2;
    }
}
```

Health Check Issues

- Config works at reload, but fails at runtime because health checks fail

```
location / {
    proxy_set_header Host $host;
    proxy_pass http://test;
    health_check;
}
```

Fails if back end requires host header to be set to a particular value – health_check sent by NGINX

Health Check Issues

- Config fails with “health check not allowed here” because `$upstream` is a runtime variable – not set

```
location / {
    proxy_pass http://$upstream;
    health_check;
}
```

Best Practice – Named Location

```
location @healthcheck_test {
    internal;
    proxy_set_header Host www.example.com;
    proxy_pass http://test;
    health_check;
    proxy_read_timeout 2s;
}
```

Module Summary

- NGINX provides a default (passive) health check
  - Fail timeout
  - Maximum number of fails
- NGINX Plus provides active health checks
  - Intervals – time between requests
  - Fails
  - Passes
  - URI
- Best practice - use named locations with health checks

Knowledge Check

Which of the following are active health check parameters?

- ☒ max_fails
- ☑ fails
- ☑ passes
- ☒ url
- ☑ uri
- ☒ server_ok
- ☑ interval

Knowledge Check

Which of the following are valid match block condition options?

- ☒ `location /health_check match=conditions;`
- ☑ `status ! 400 402 404;`
- ☑ `header Content-Type = text/html;`
- ☑ `body ~ “Maintenance”`
- ☒ `health_check match=server_ok;`

Knowledge Check

True or False? The NGINX Plus active health check allows you to set the URI to be checked.

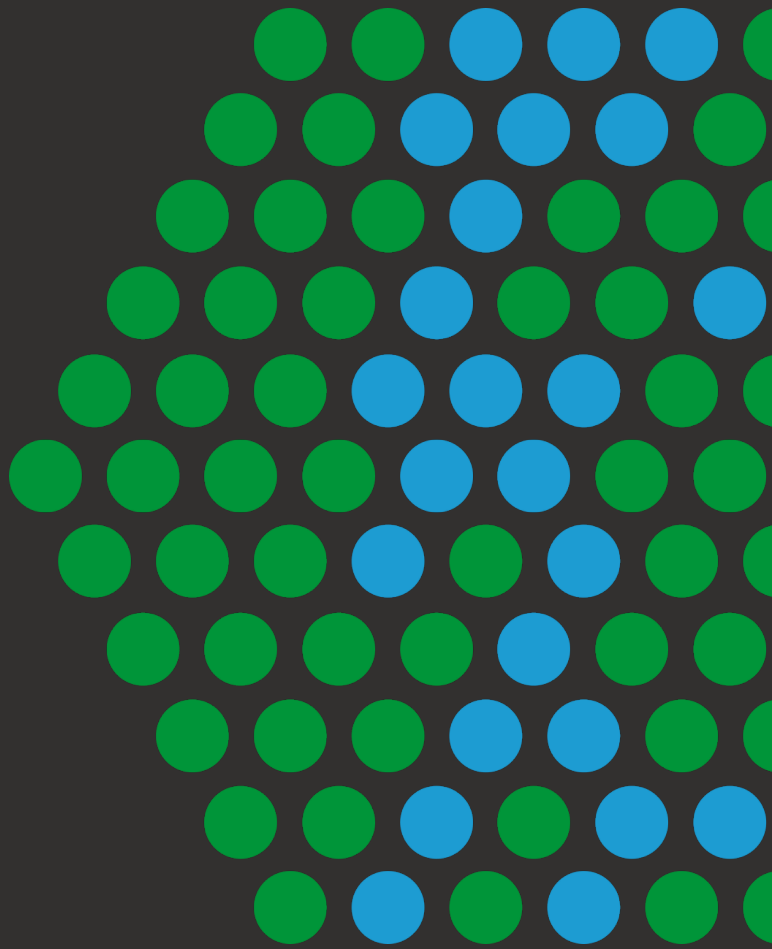
- ☑ True
- ☒ False

Lab 13 – Health Checks

NGINX
Part of F5

Module 14

Caching

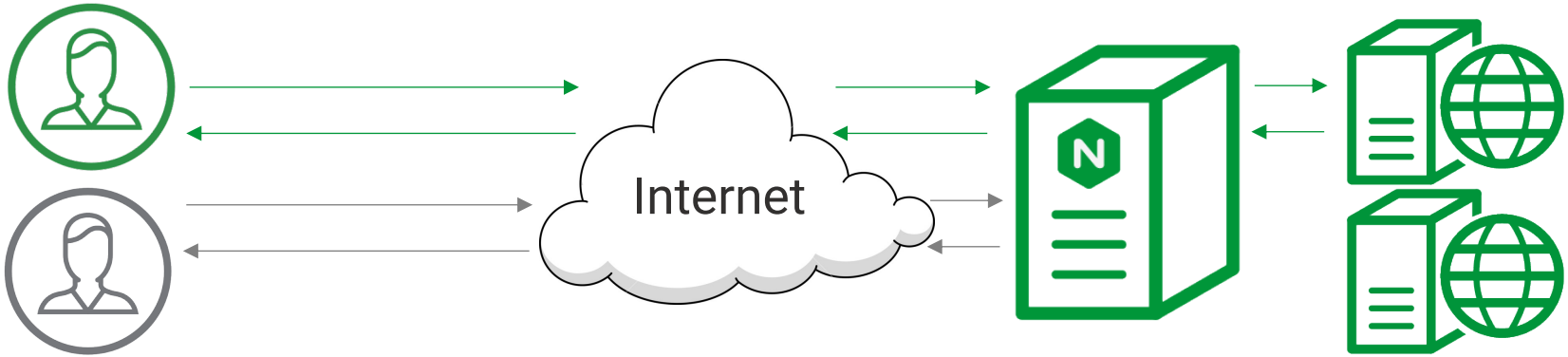


Module 14 Objectives

- Set up a reverse proxy cache for upstream servers
- Identify how the hybrid persistent cache model works

Types of Caching

- Browser cache
- Content delivery network (CDN)
- Reverse proxy cache

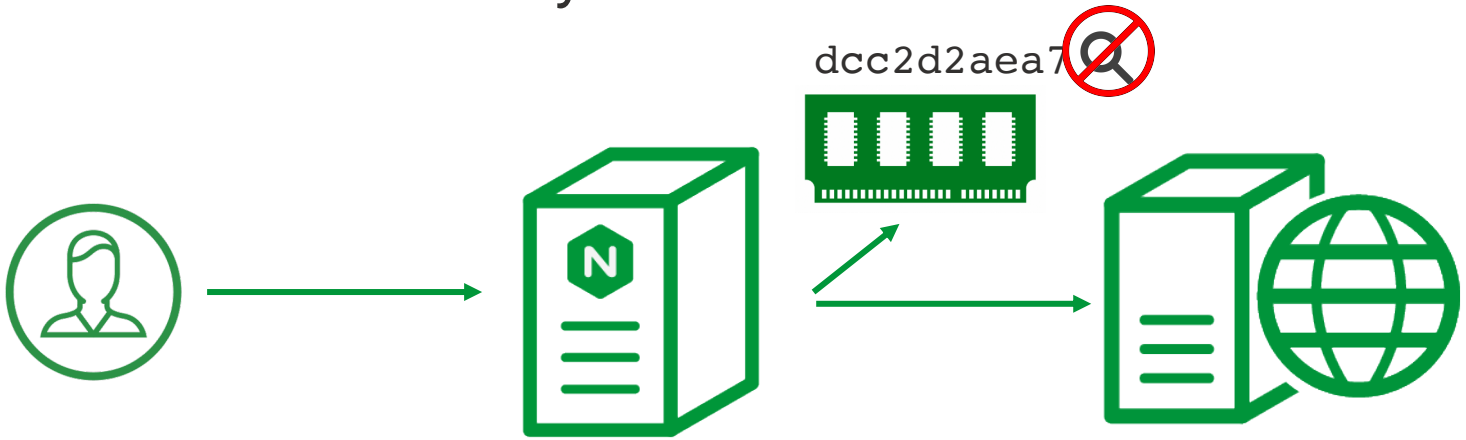


Caching Process

- Client request
- NGINX reverse proxy receives

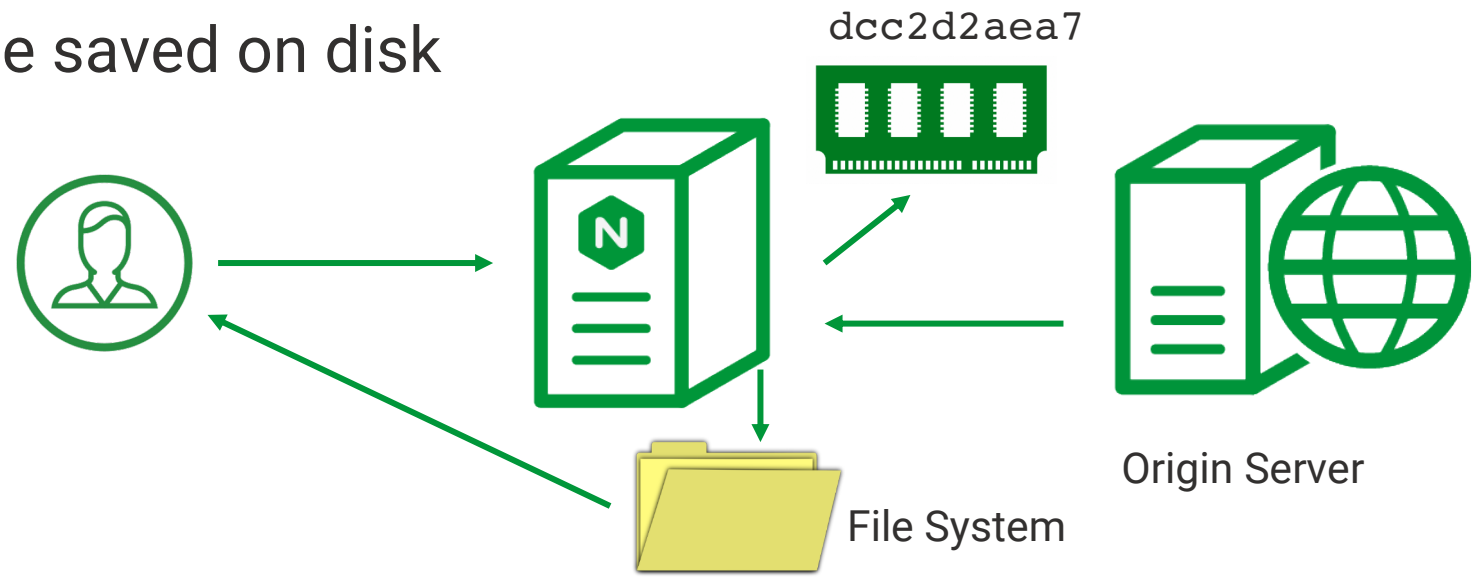
Caching Process

- NGINX generates MD5 hash
- NGINX checks memory for hash



Caching Process

- Origin server sends response
- Hash saved in memory
- File saved on disk



Hybrid Persistent Cache

- Content cached on disk
- Hash cached in RAM

proxy_cache_path Syntax

- Defines cache location and directory level
- Defines memory zone and size (for hash / keys)
- How long to cache information
- How much disk to use to cache content
- Directives
  - `proxy_cache_path`
  - `proxy_cache`

proxy_cache_path Example

```
http {
    proxy_cache_path /data/nginx/cache
    levels=1:2 keys_zone=images:20m;
}
```

Hash Key

File name:

/data/nginx/cache/c/29/b7f54b2df7773722d382f4809d65029c

Additional Parameters

Parameter	Use	Default
`inactive=20s`	How long to cache	10 minutes
`max_size=5m`	Disk size for content	
`manager_files=50`	Files deleted when max_size is exceeded*	100
`purger=on|off*`	Purges entries matching wildcard key	

*NGINX Plus only feature

Additional Caching Directives

- **`proxy_cache_key`** *`string;`*
- **`proxy_cache_valid`** *`[code] time;`*
- **`add_header`** *`header value;`*

Module Summary

- Required caching directives
  - `proxy_cache_path` – HTTP context
  - `proxy_cache` – location context
- Directory of cache set by levels parameter
- Shared memory zone required
- Optional parameters
  - Hash key
  - Content disk size
  - How long to cache content

http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_cache_key

Knowledge Check

True or False? The purpose of caching is to put the proxy server closer to the content.

- ☒ True
- ☑ False

Knowledge Check

With this configuration, and the hash key listed, where will the cached content be stored?

```
http {
    proxy_cache_path /data/nginx/cache
    levels=2:2:1 keys_zone=images:20m;
}
Hash:   b7f54b2df7773722d382f4809d65034d
```

- ☑ /data/nginx/cache/4d/03/5/b7f54b2df7773722d382f4809d65034d
- ☒ /data/nginx/cache/d/34/0/b7f54b2df7773722d382f4809d65034d
- ☒ /data/nginx/cache/50/34/4/b7f54b2df7773722d382f4809d65034d

Knowledge Check

Which of the following parameters indicates the size of the shared memory zone?

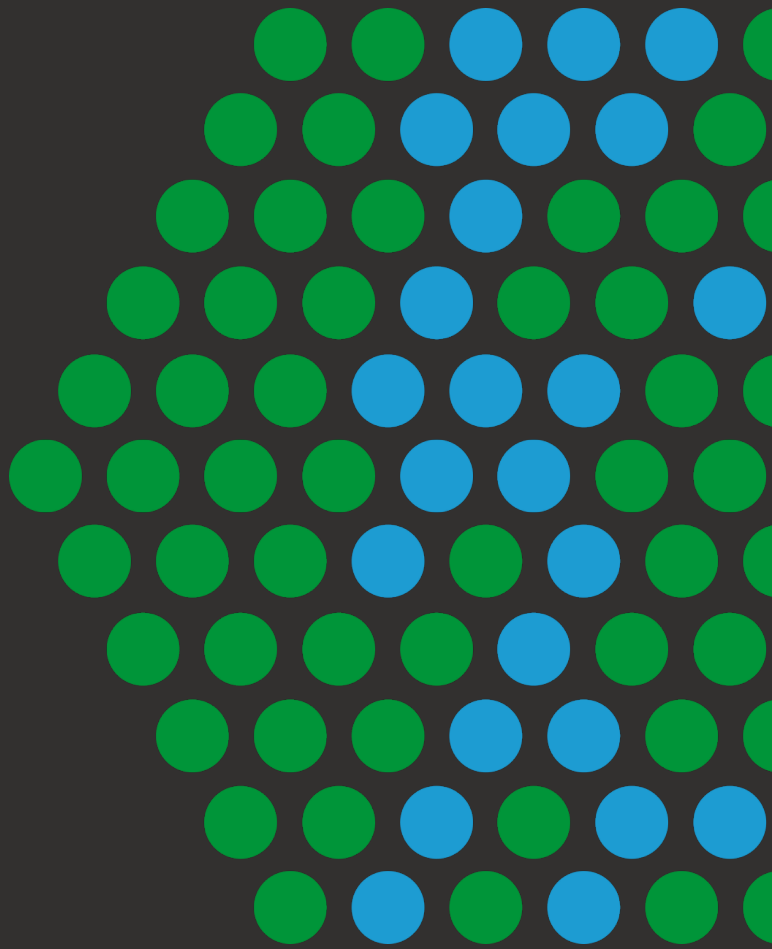
- ☒ `levels=1:2`
- ☒ `inactive=20m`
- ☒ `manager_files=50`
- ☒ `max_size=5m`
- ☑ `keys_zone=images:20m`

Lab – Caching

NGINX
Part of F5

Module 15

Encrypting Web Traffic



Module 15 Objectives

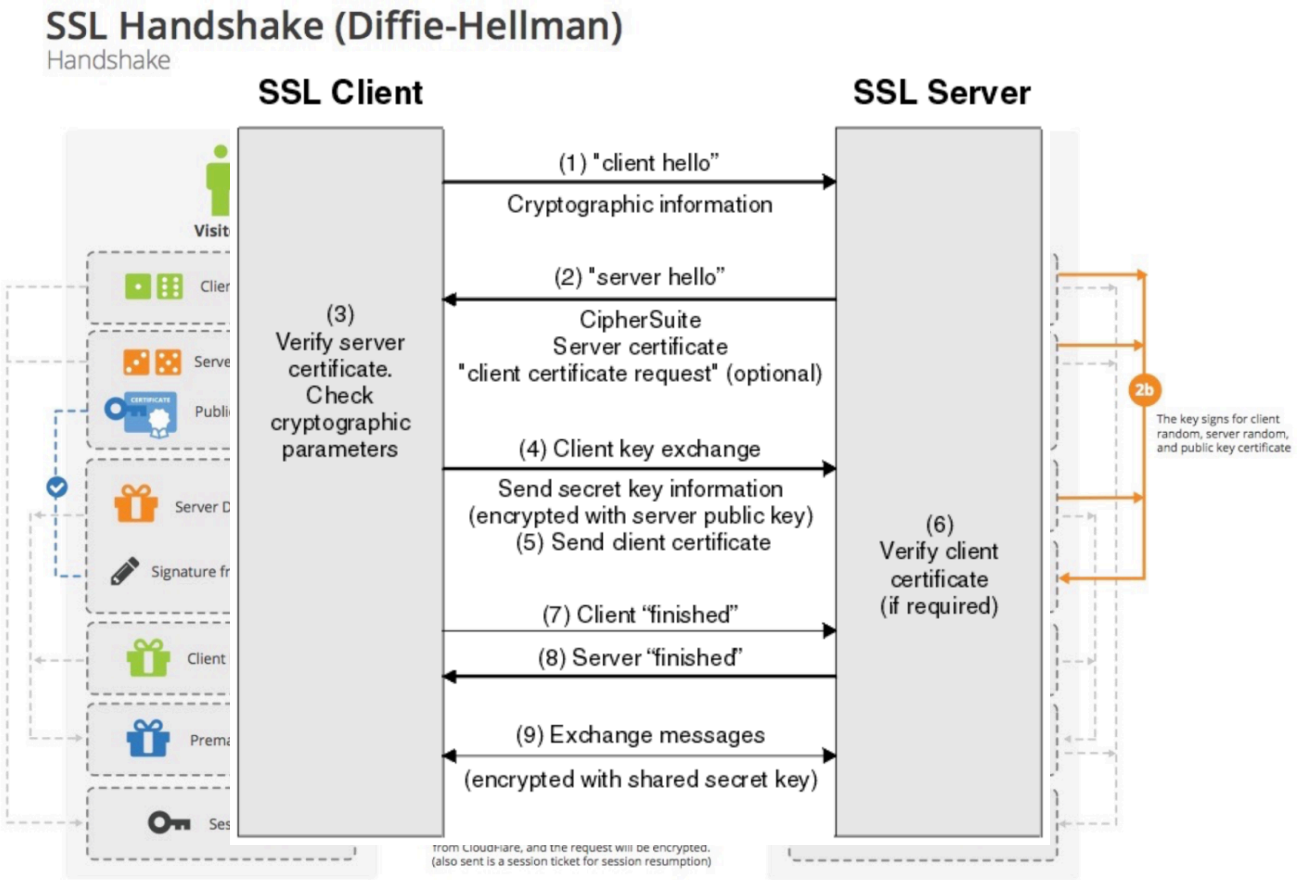
- Set up HTTPS access
- Set up encryption levels on your proxy server
- Test the security of your server against ssllabs.com

Qualys. SSL Labs

HTTPS

- HTTP + Security
- SSL / TLS
- Data encryption
- Server authentication
- Key negotiation in the handshake

Onerous TLS Handshake



Using HTTPS

- User privacy –required across the Web now
- Front older services
- Complexity
- CPU cost

Setting up HTTPS

- `openssl` library
  - Automatically comes with NGINX Plus
  - Must add if compiling NGINX OSS
- NGINX has directives to encrypt both ends

Building SSL Configuration

- Enable `ssl` on the `listen` directive
- Specify the `ssl_certificate`
- Specify the `ssl_certificate_key`

```
server {
    listen 443 ssl;
    root /home/ubuntu/public_html;
    ssl_certificate /etc/nginx/ssl/nginx.crt;
    ssl_certificate_key /etc/nginx/ssl/nginx.key;
}
```

Building SSL Configuration

```
server {
    listen 443 ssl;
    root /home/ubuntu/public_html;
    ssl_certificate /etc/nginx/ssl/nginx.crt;
    ssl_certificate_key /etc/nginx/ssl/nginx.key;

    ssl_dhparam ssl/dhparam.pem;
    ssl_protocols TLSv1.1 TLSv1.2;
    ssl_ciphers AES256+EECDH:AES256+EDH:!aNULL;
    ssl_prefer_server_ciphers on;
}
```

Add Headers

- Force browsers to communicate over HTTPS using `add_header` directive with HTTP Strict Transport Security (HSTS)
- No sniffing
- No IFrames

```
add_header Strict-Transport-Security: max-age=31536000
add_header X-Content-Type-Options nosniff;
add_header X-Frame-Options DENY;
```

Additional (optional) SSL Directives

- `ssl_stapling`
- `ssl_dhparam`
- `ssl_session_cache`

https://blog.cloudflare.com/tls-session-resumption-full-speed-and-secure/

Syntax to Force HTTPS

```
#server 1
server {
    listen 80;
    return 301 https://$host$request_uri;
}
# server 2
listen 443 ssl;
ssl_certificate /etc/nginx/ssl/server.crt;
ssl_certificate_key /etc/nginx/ssl/server.key
```

Additional Best Practices for Security

What to Do	Related Code
Don’t allow NGINX versions to be discovered	`server_tokens off;`
Turn off backend engine headers	`X-Powered-By`
Change client-side error pages	`error_page` *`code uri;`*
Encrypt everything	ssl & proxy_ssl

Note: Test on SSL Labs Web site: https://www.ssllabs.com/

Module Summary

- NGINX provides TLS / SSL termination
  - Front end with `ssl` parameter
  - Back end with `proxy_ssl`
- Build SSL Config with
  - `ssl` on listen directive
  - `ssl_certificate` and `ssl_certificate_key`
  - `ssl_protocols`
  - `ssl_ciphers`
  - `ssl_prefer_server_ciphers`
  - `add_header`

Knowledge Check

True or False? NGINX requires the openssl library be installed in order to set up HTTPS encryption.

- ☑ True
- ☒ False

Knowledge Check

With this configuration, what is NGINX behavior?

```
ssl_prefer_server_ciphers on;
```

- ☑ NGINX prefers the order of SSL ciphers listed on the server block but negotiates with the browser to decide which to use.
- ☒ NGINX turns on the ability to use SSL ciphers.
- ☒ NGINX uses its list of SSL ciphers only.

Knowledge Check

Which of the following are NGINX security best practices?

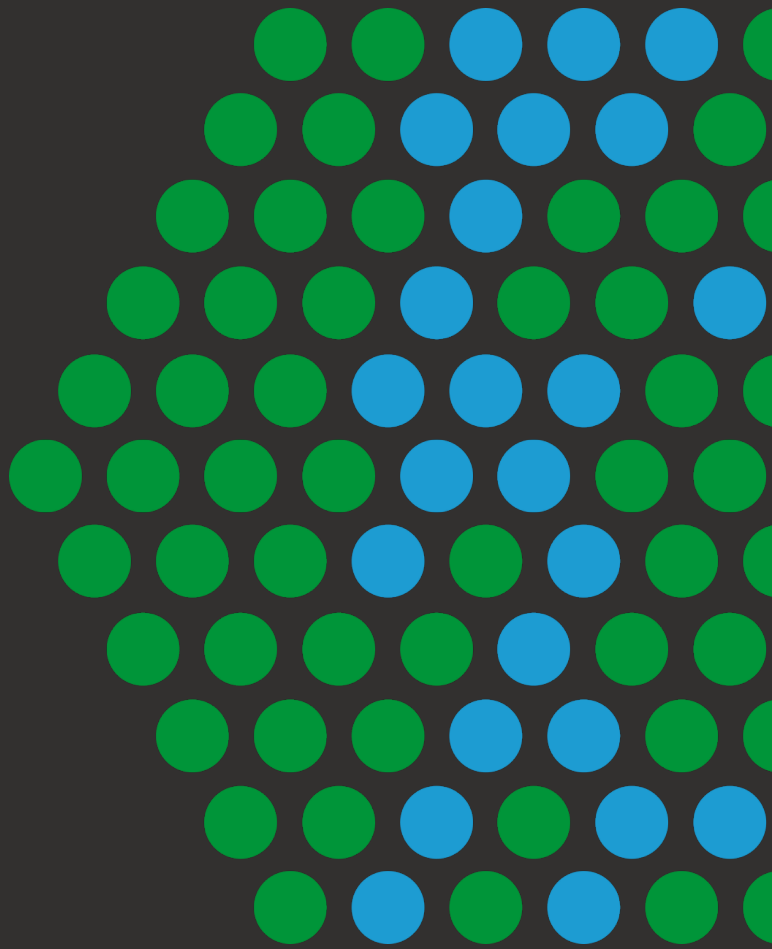
- [x] `ssl_prefer_server_ciphers on;`
- [x] `add_header Strict-Transport-Security: max-age=31536000;`
- [ ] `add_header X-Frame-Options OFF;`

Lab – Encrypting Web Traffic

NGINX
Part of F5

Module 16

Restricting Access



Module 16 Objectives

- Determine multiple methods to restrict access
  - `allow` / `deny` directives
  - Authentication
  - Limiting access
    - Limit Rates
    - Limit connections

allow and deny Directives

- Used together in http, server, and location contexts
- `allow` – provides access
- `deny` – prevents access

```
location /api {
    allow 125.2.2.0/16;
    deny all;
}
```

Authentication

- Basic – uses password file
- Based on Sub request – redirects to location
  - Location proxies to auth server
  - Set a variable from the authorization request

```
auth_basic realm;
auth_basic_user_file file_name;

auth_request uri;
auth_request_set $variable value;
```

Limiting Access

- Connections
- Request rate
- Download speeds

Limiting Connections

- Limit connections per IP or per server
- Define shared memory zone and reference in location

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

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

Limiting Request Rate

- Limit number of requests per second
- Define shared memory zone and reference in location

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

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

Limiting Download Rate

- Allows client to use max download speed to build buffer
- Limit download based on video bitrate – allow smooth playback

```
    server { . . .
        location /media {
            limit_rate_after 500k;
        }   limit_rate 50k;
}       }
}
```

Global Rate Limiting

- Shares runtime states across nodes in a cluster
- Master/Primary node config not required with
  - Network connectivity between all nodes
  - Synchronized clocks
  - zone_sync configuration

```
resolver 10.0.0.53 valid=20s;

    server { . . .
        listen 9000;
            zone_sync;
            zone_sync_server cluster.com:9000 resolve;
        }
}
```

Module Summary

- NGINX restricts access with
  - `allow` and `deny` directives
  - authentication methods
  - Connection and rate limiting methods
    - Connections
    - Request rate
    - Download rate
    - Global rate limiting for clusters

Knowledge Check

True or False? NGINX global rate limiting functions within clusters.

- ☑ True
- ☒ False

Knowledge Check

With this configuration, what is the NGINX behavior?

```
server { ...
    location /private/ {
        auth_request /auth;
        auth_request_set $auth_status $upstream_status;
}
    location = /auth {
        internal;
        proxy_pass http://ldap-server;
        proxy_set_header <header> <$variable>;
        } } }
```

NGINX authenticates with a password file.

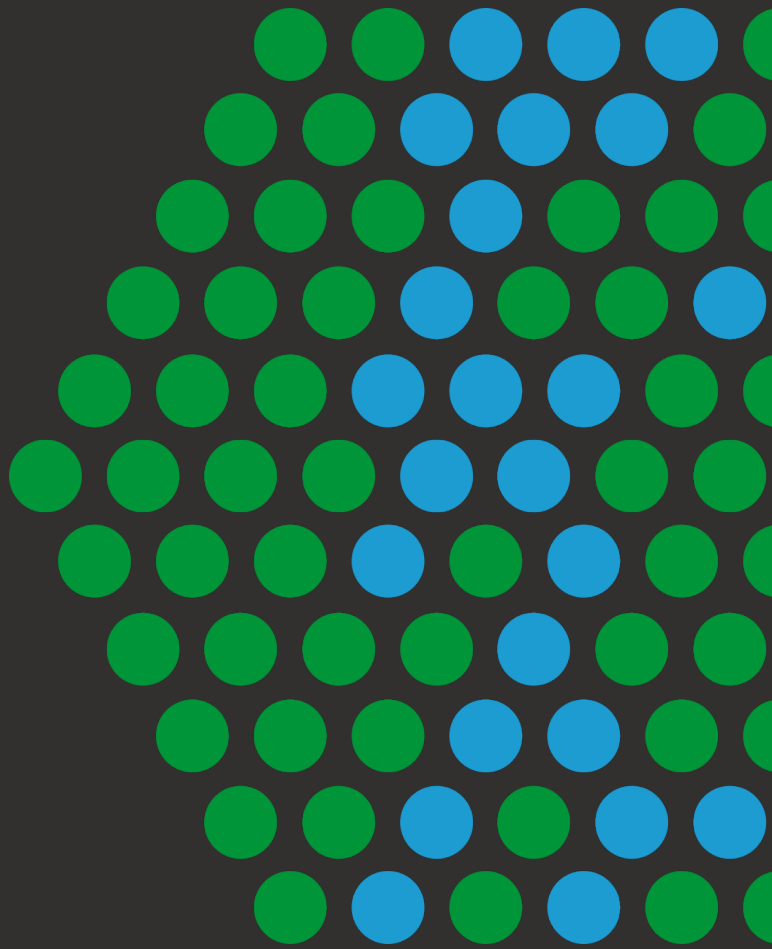
NGINX passes authentication to an authentication server.

Knowledge Check

With this configuration ,what is the NGINX behavior?

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

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

- [x] NGINX limits the number of connections to 5 per IP address
- [ ] NGINX limits the rate of download to 10MB/s

NGINX
Part of F5

Thank You!