Nginx Research (Task 2)

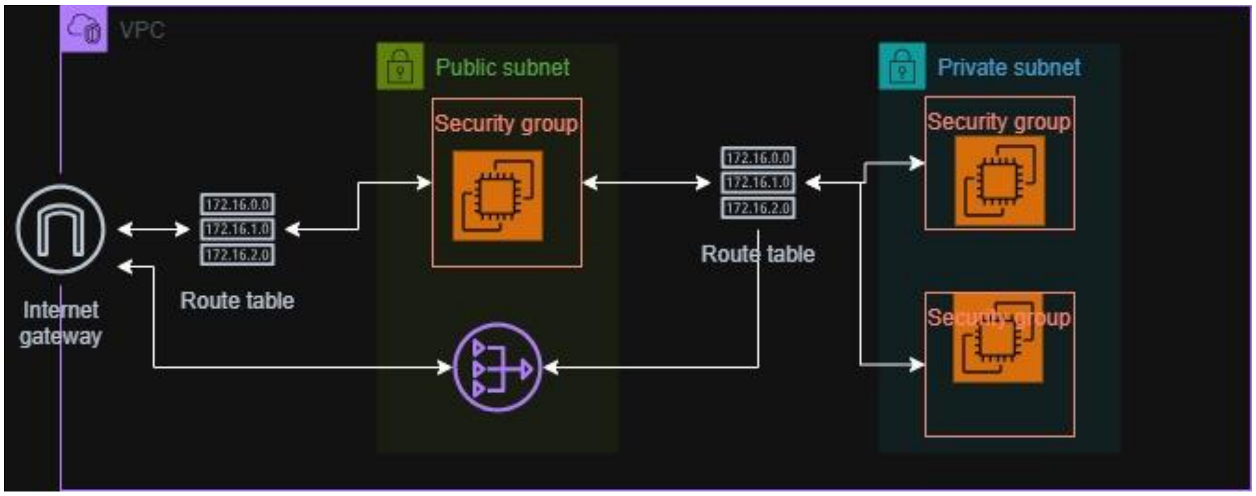
Zaeem Attique Ashar

Cloud Intern

Task Description:

This task is about researching one of the most popular webservers called Nginx. After covering the basics, you will take a deeper dive into various services provided by Nginx such as setting up a reverse proxy and a load balancer, Installing SSH/TLS certificates, integrating PHP using PHP-FPM and monitoring/troubleshooting using logs. This task will be more focused on researching and configuring these services provided by Nginx.

Architecture Diagram Used for Testing Configurations:



Task2.0: Deploying the infrastructure for testing the research.

- VPC Configuration:
  - Type VPC in the search bar and head over to the VPC Dashboard.
  - Start creating a new VPC by clicking on the yellow Create VPC button.
  - Select VPC only option.
  - Name your VPC (Task2-VPC-Zaeem).
  - Select the IPv4 CIDR manual input and enter 10.0.0.0/16 in the field below.
  - Select the 'No IPv6 CIDR block' option next.
  - Click on the Create VPC button in the bottom right corner.
- Subnets Configuration:
  - From the left panel in the VPC Dashboard, head over to the subnets tab.
  - Start creating a new subnet by clicking on the Create Subnet button in the top right corner.
  - Select the VPC created in the last step from the drop-down menu.
  - Name the first Subnet (Task2-PublicSN-Zaeem).
  - Select the availability zone desired (us-west-2a).
  - The IPv4 CIDR block will be automatically selected as we have only one CIDR block defined in the VPC.
  - Enter the desired IPv4 subnet CIDR block for the subnet (10.0.1.0/24).
  - Use the Add New Subnet button at the bottom left to create another subnet.
  - Assign the private subnet with a name.
  - Select the same availability zone selected for the previous subnet.
  - The IPv4 VPC CIDR block will remain the same.
  - Assign an IPv4 subnet CIDR Block (10.0.2.0/24).
  - Use the Create Subnet button in the bottom right corner to create the 2 subnets.

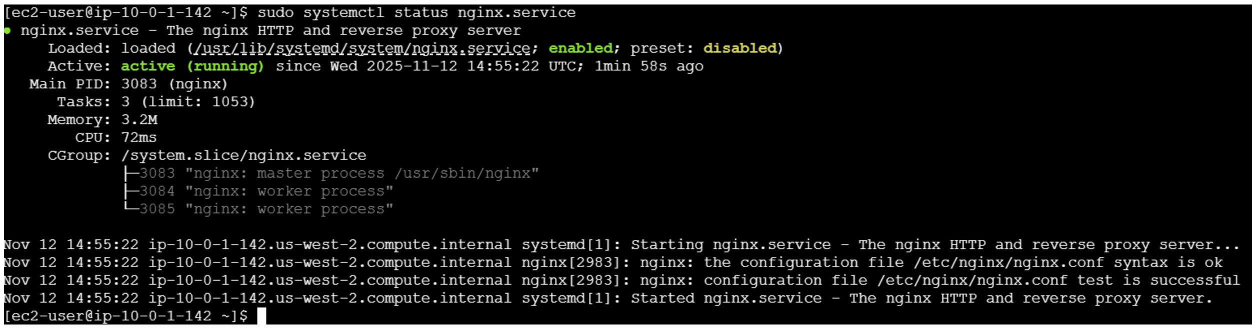
- Route Table Configuration:
    - Use the Create Route Table button in the top right corner.
    - Name the Route Table (Task2-PublicRT-Zaeem).
    - Select the VPC previously created from the dropdown menu.
    - Use the Create Route Table in the bottom right corner to create.
    - Edit the inbound route and add: Destination 0.0.0.0/0, Target IGW.
    - Open the route table by clicking on the ID and click on the Actions button on the top right corner, then select the Edit Subnet Association option.
    - Now from the list, select the public subnet that we previously created.
    - Carry out the same steps to create the Private Route Table (Task2-PrivateRT-Zaeem) in the same VPC.
    - Edit the inbound route and add: Destination 0.0.0.0/0, Target NGW.
    - Open the route table by clicking on the ID and click on the Actions button on the top right corner, then select the Edit Subnet Association option.
    - Now from the list, select the private subnet that we previously created.
- Internet Gateway Configuration:
    - Go to the Internet Gateway tab in the VPC Dashboard.
    - Use the Create Internet Gateway button in the top right corner to start.
    - Name the Internet Gateway.
    - Click on the Create internet gateway button in the bottom right corner.
    - Now click on the Actions button and select Attach to VPC from the dropdown menu.
    - Select the VPC created previously and click the Attach internet gateway button in the bottom right corner.
- NAT Gateway Configuration:
    - Go to the NAT gateway tab in the left panel in the VPC Dashboard.
    - Name the NAT gateway.
    - Select the public subnet previously created.
    - Select connectivity type as public.
    - Click on the Allocate Elastic IP button to get an EIP for the NGW.
    - Click on the Create NAT gateway button in the bottom right corner.
- EC2 Instances (Public) Configuration:
    - Go to the EC2 Dashboard from the search bar.
    - Go to the instances tab from the left panel.
    - Click on the launch instances button at the top right corner.
    - Name your Public EC2 Instance (Task2-PublicInstance-Zaeem).
    - Select your AMI (Amazon Linux 2023).
    - Select instance type t3.micro.
    - Click on the edit button in network settings.
    - Select the VPC created for this task.

  - Select the public subnet.
  - Select ‘enable’ under the auto-assign public IP button.
  - Select create security group under firewall settings.
  - Add the following rules to the Security Group’s Inbound Traffic:
    - Type: SSH, Source: 0.0.0.0/0
    - Type: HTTP, Source: 0.0.0.0/0
  - Under configure storage: 1x 8 GiB gp3 for root volume.
- EC2 Instances (Private) Configuration:
  - Go to the EC2 Dashboard from the search bar.
  - Go to the instances tab from the left panel.
  - Click on the launch instances button at the top right corner.
  - Name your Public EC2 Instance (Task2-PrivateInstance-Zaeem).
  - Select your AMI (Amazon Linux 2023).
  - Select instance type t3.micro.
  - Click on the edit button in network settings.
  - Select the VPC created for this task.
  - Select the private subnet.
  - Select create security group under firewall settings.
  - Add the following rules to the Security Group (Outbound Traffic):
    - Destination: 0.0.0.0/0, Target: NGW
  - Under configure storage: 1x 8 GiB gp3 for root volume.

Task2.1: Nginx Introduction and Installation

Nginx is a popular web server that provides various other features as well as Reverse Proxy Server, Load Balancer, HTTP Cache and more.

- SSH into the public EC2 Instance from the Instances panel by clicking on the checkbox and using the connect button at the top.
- After accessing the shell, enter the following commands:
    - sudo dnf update –y
    - sudo dnf install nginx
    - Sudo systemctl enable nginx.service
    - Sudo systemctl status nginx.service

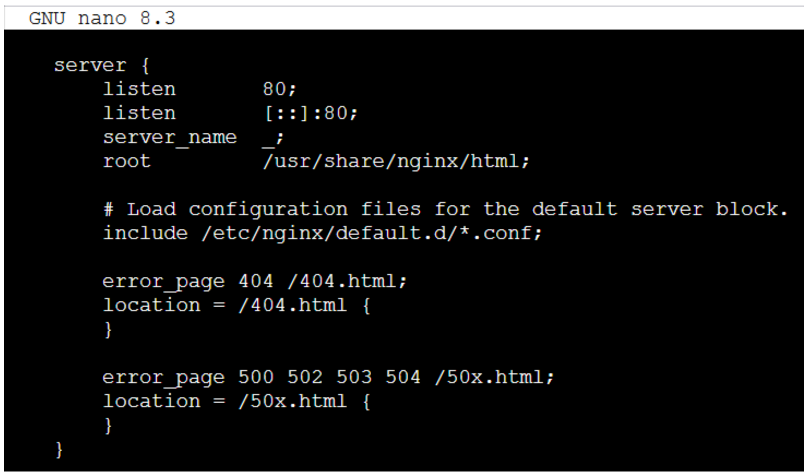


```
[ec2-user@ip-10-0-1-142 ~]$ sudo systemctl status nginx.service
● nginx.service - The nginx HTTP and reverse proxy server
     Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; preset: disabled)
     Active: active (running) since Wed 2025-11-12 14:55:22 UTC; 1min 58s ago
   Main PID: 3083 (nginx)
      Tasks: 3 (limit: 1053)
     Memory: 3.2M
        CPU: 72ms
     CGroup: /system.slice/nginx.service
             ├─3083 "nginx: master process /usr/sbin/nginx"
             ├─3084 "nginx: worker process"
             └─3085 "nginx: worker process"

Nov 12 14:55:22 ip-10-0-1-142.us-west-2.compute.internal systemd[1]: Starting nginx.service - The nginx HTTP and reverse proxy server...
Nov 12 14:55:22 ip-10-0-1-142.us-west-2.compute.internal nginx[2983]: nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
Nov 12 14:55:22 ip-10-0-1-142.us-west-2.compute.internal nginx[2983]: nginx: configuration file /etc/nginx/nginx.conf test is successful
Nov 12 14:55:22 ip-10-0-1-142.us-west-2.compute.internal systemd[1]: Started nginx.service - The nginx HTTP and reverse proxy server.
[ec2-user@ip-10-0-1-142 ~]$
```

Task2.1: Serve static content using root and index directives

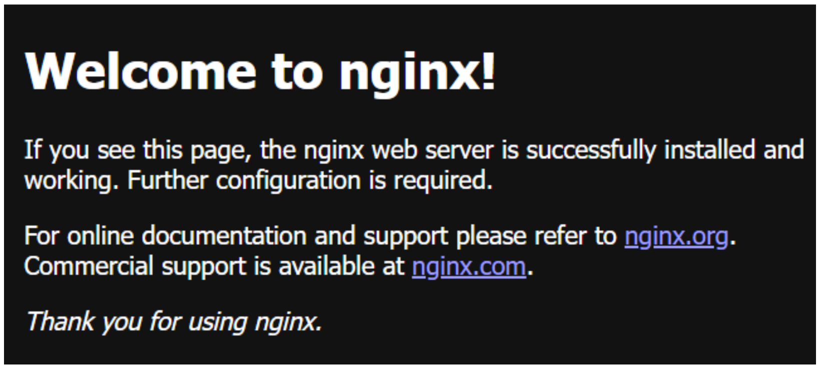
Nginx configuration is located in the /etc/nginx/nginx.conf file in RedHat distributions and at /etc/nginx/sites-available/default in debian distributions.

We have a default static webpage located at /usr/share/nginx/html/index.html which is configured in the nginx.conf file out of the box.



```
 GNU nano 8.3

    server {
        listen       80;
        listen       [::]:80;
        server_name  _;
        root         /usr/share/nginx/html;

        # Load configuration files for the default server block.
        include /etc/nginx/default.d/*.conf;

        error_page 404 /404.html;
        location = /404.html {
        }

        error_page 500 502 503 504 /50x.html;
        location = /50x.html {
        }
    }
```

We can access this page by entering the Public IP of the instance in the browser.



Welcome to nginx!

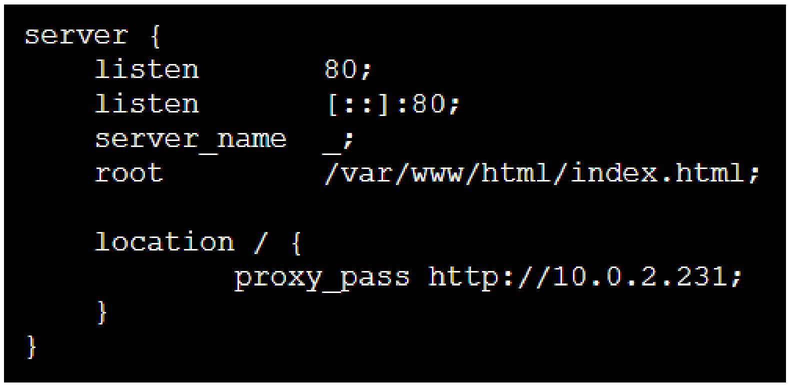
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Task2.2: Configure reverse proxy with proxy_pass and custom headers

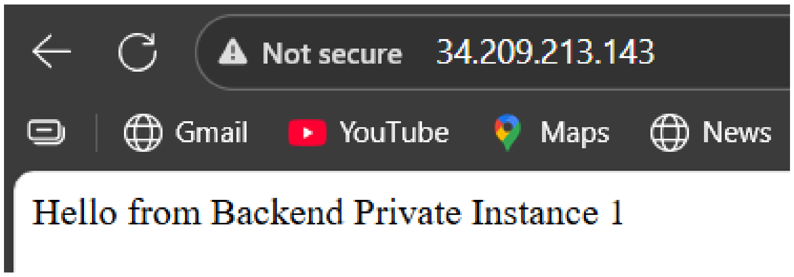
The reverse proxy server sits between the client and the backend server. It allows us to not expose the IP/DNS of our web server directly to the client. To setup the reverse proxy server in nginx we use the proxy_pass directive in the nginx.conf file. Below is a test configuration of a reverse proxy.



```
server {
    listen       80;
    listen       [::]:80;
    server_name  _;
    root         /var/www/html/index.html;

    location / {
            proxy_pass http://10.0.2.231;
    }
}
```

After making the changes in the config file we must always run “sudo systemctl reload nginx” to apply the changes we made.

Now when we enter the IP of our public instance in the browser, instead of serving the local page it returns the page sent in response by one of the backend server.

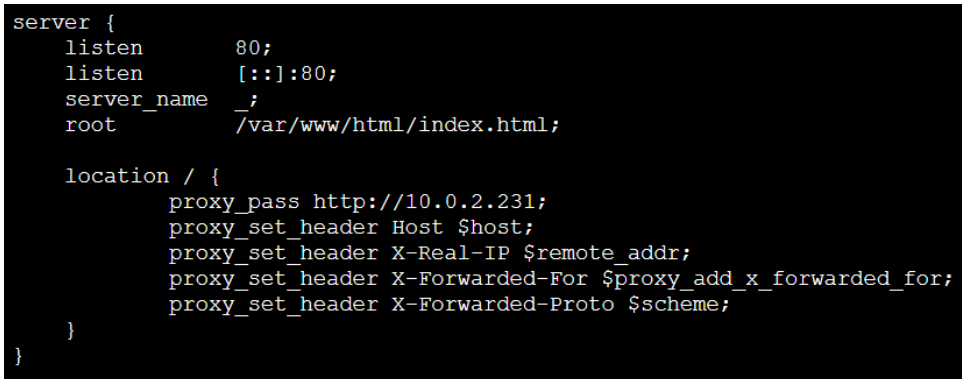


Hello from Backend Private Instance 1

Headers are information that is forwarded to the backend server for further functionality and security. Here are some common headers that are used:

- proxy_set_header Host $host; (Passes the original domain (Host header) to the backend).
- proxy_set_header X-Real-IP $remote_addr; (Send the client's real IP instead).
- proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for; (Maintain the list of client IPs if there are multiple proxies).
- proxy_set_header X-Forwarded-Proto $scheme; (Tells backend whether client used HTTP or HTTPS).

Here is a sample of these headers configured in the nginx.conf file.



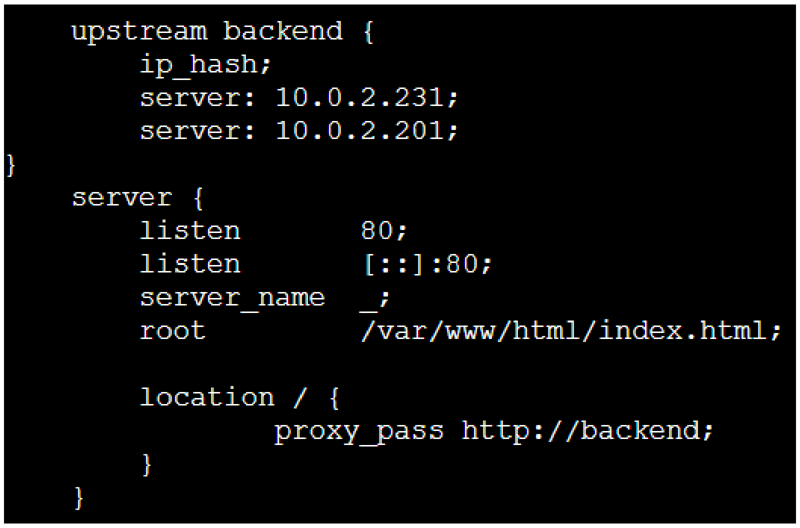
```
server {
    listen       80;
    listen       [::]:80;
    server_name  _;
    root         /var/www/html/index.html;

    location / {
            proxy_pass http://10.0.2.231;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

Task2.3: Set up load balancing with round robin, least_conn, and ip_hash

Another feature of Nginx is load balancing. If we have multiple backend servers, we can distribute the traffic according to various different distribution techniques such as round robin. Below are a few popular load distribution algorithms:

- **Round Robin:** Distribute traffic one after the other. Loop back to the first server after the last one is done. Useful for when the servers have identical resources. It doesn't need to be configured as it is the default load balancing algo.
- **Least Conn:** Traffic is forwarded to the server that has the least active connections. This helps distribute the load evenly when the connection TTL varies.
- **IP Hash:** Requests from the same client's IP are always sent to the same backend server. This is how sticky sessions work to preserve user sessions.
- **Weighted Round Robin:** Traffic is distributed in a sequence, one after the other but with weight. Meaning that the server assigned more weight will get more requests.
- **Backup Servers:** Traffic is only distributed to backup servers when no other server responds (is down).

To use the load balancing feature, define the upstream block outside the server block. Under this block, set the name of the distribution algorithm. We can then use the name of the upstream block ('backend' in this example) in the proxy pass directive. Reload the nginx.service after the changes are made in the config file.



```
    upstream backend {
        ip_hash;
        server: 10.0.2.231;
        server: 10.0.2.201;
}
    server {
        listen        80;
        listen        [::]:80;
        server_name   _;
        root          /var/www/html/index.html;

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

Task2.4: Enable SSL/TLS using self-signed certificates

TLS Certificates are cryptographic certificates that verify the identity of the website. These certificates are signed by a CA that proves the website is authentic. When a client starts a connection with a webserver, it sends the certificate in response to confirm that the client is talking to the authentic website instead of a middleman or a fake website. We can also generate self-signed certificates that will encrypt the traffic but would show secure connection on the browser. For this purpose, we need to install the openssl tool in linux.

Use the command below to generate the certificate and the key:

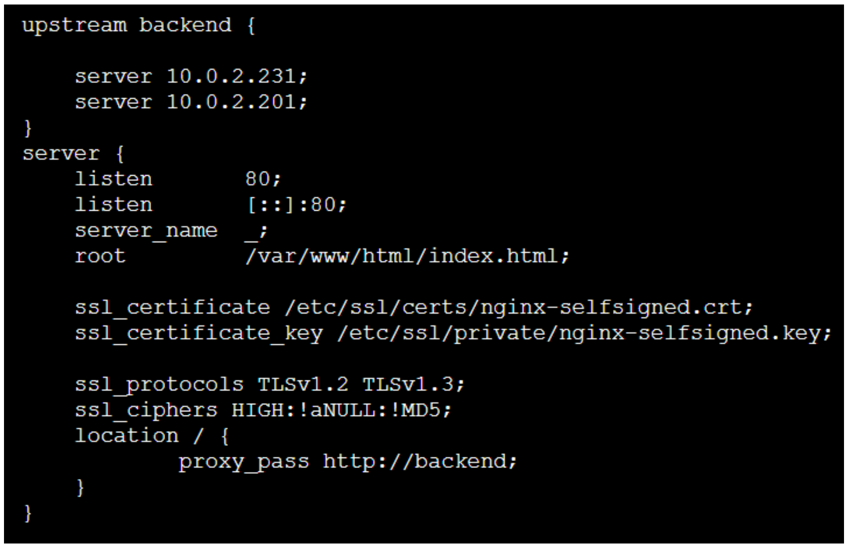
sudo openssl req -x509 -newkey rsa:2048 -keyout /etc/ssl/private/nginx-selfsigned.key

-out /etc/ssl/certs/nginx-selfsigned.crt -days 365 –nodes

Command breakdown:

- **Req**: Create a certificate signing request (CSR)
- **-x509**: Create a self-signed certificate instead of a CSR
- -**newkey rsa**:2048: Generate a new 2048-bit RSA key pair
- -**keyout**: Path to save the private key
- -**out**: Path to save the certificate
- -**days 365**: Valid for 1 year
- -**nodes**: Skip password protection (so Nginx can start without manual password input)

Now we can use the paths to the generated key in the nginx.conf file to make use of it.



```
upstream backend {

    server 10.0.2.231;
    server 10.0.2.201;
}
server {
    listen       80;
    listen       [::]:80;
    server_name  _;
    root         /var/www/html/index.html;

    ssl_certificate /etc/ssl/certs/nginx-selfsigned.crt;
    ssl_certificate_key /etc/ssl/private/nginx-selfsigned.key;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;
    location / {
            proxy_pass http://backend;
    }
}
```

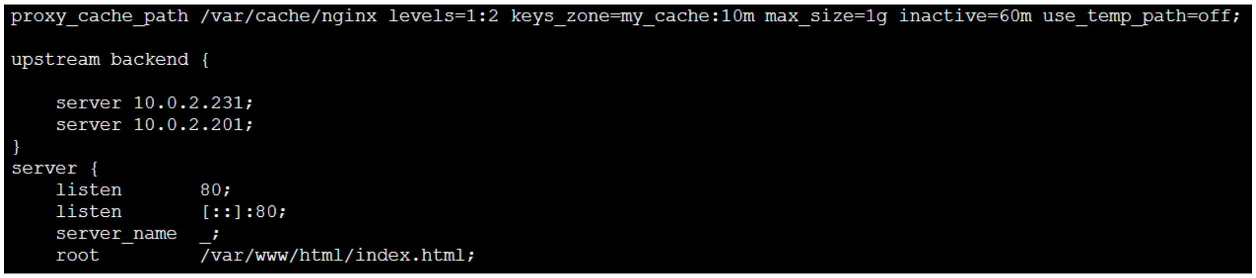
Task2.5: Implement caching with proxy_cache and validate it

Nginx provides yet another useful feature called a caching server which stores the frequently accessed static content in its local storage. When a client requests static content, the caching server first checks the local storage. If the content is found, it sends it back to the client directly without forwarding the request to the backend server. If not found or is expired, the request is passed to the backend server, and the response is then cached for future use. This greatly reduces the backend server resource usage.

To set up caching in nginx, we first need to create a directory to store the cache files. Use the following commands:

- *sudo mkdir -p /var/cache/nginx*
- *sudo chown -R nginx:nginx /var/cache/nginx*

Next, we need to add the path to the created directory **outside** the server block:

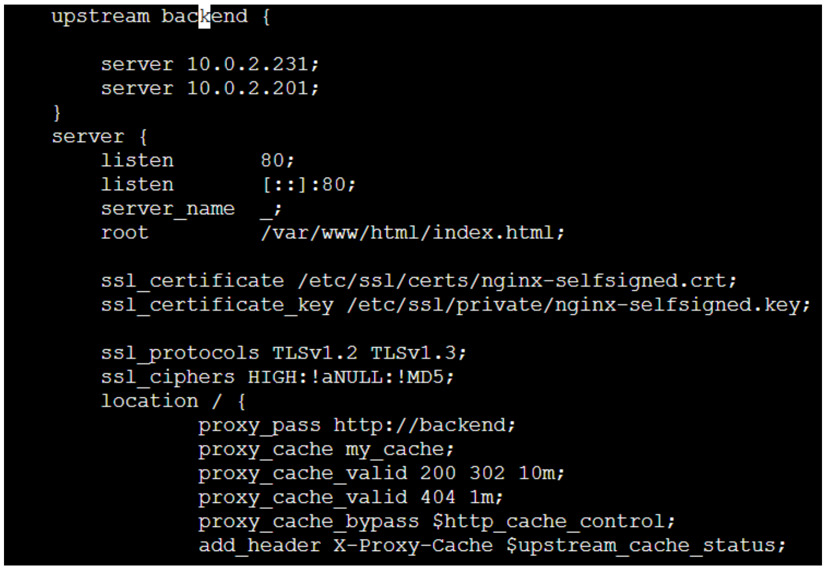


```
proxy_cache_path /var/cache/nginx levels=1:2 keys_zone=my_cache:10m max_size=1g inactive=60m use_temp_path=off;

upstream backend {

    server 10.0.2.231;
    server 10.0.2.201;
}
server {
    listen       80;
    listen       [::]:80;
    server_name  _;
    root         /var/www/html/index.html;
```

Explanation of the parameters is as follows:

- /var/cache/nginx: Location to store cache files
- levels=1:2: Directory structure depth (e.g., /a/1a/ style folders)
- keys_zone=my_cache:10m: Creates a named cache zone in memory (10MB index of cache keys)
- max_size=1g: Maximum cache size (older files get purged)
- inactive=60m: Items not accessed for 60 minutes are removed
- use_temp_path=off: Writes directly to the cache directory (improves performance)

Now we need to configure the server to use the cashing:



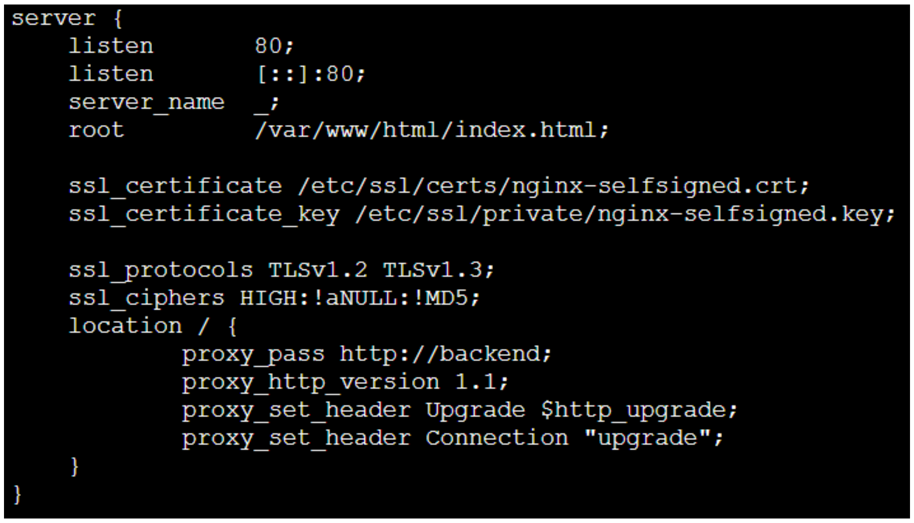
```
upstream backend {

    server 10.0.2.231;
    server 10.0.2.201;
}
server {
    listen       80;
    listen       [::]:80;
    server_name  _;
    root         /var/www/html/index.html;

    ssl_certificate /etc/ssl/certs/nginx-selfsigned.crt;
    ssl_certificate_key /etc/ssl/private/nginx-selfsigned.key;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;
    location / {
            proxy_pass http://backend;
            proxy_cache my_cache;
            proxy_cache_valid 200 302 10m;
            proxy_cache_valid 404 1m;
            proxy_cache_bypass $http_cache_control;
            add_header X-Proxy-Cache $upstream_cache_status;
```

Caching Directives Explained:

- proxy_cache my_cache; Enables caching using the cache zone defined earlier.
- proxy_cache_valid 200 302 10m; Cache successful (200) and redirect (302) responses for 10 minutes.
- proxy_cache_valid 404 1m; Cache 404 errors for 1 minute.
- proxy_cache_bypass $http_cache_control; Skip cache when browser sends a "no-cache" header.
- add_header X-Proxy-Cache $upstream_cache_status; Adds a header to responses showing cache status (e.g., MISS, HIT).

Task2.6: Add WebSocket support through proxy configuration

WebSocket is a newer version of HTTP protocol that supports full duplex communication between a client and a server. It allows long-lived TCP connections. The client requests a ws connection through an initial HTTP handshake that basically says that the client would like to upgrade the communication to WS. In the nginx.conf file, we need to use the appropriate directives to preserve the WS headers.



```
server {
    listen       80;
    listen       [::]:80;
    server_name  _;
    root         /var/www/html/index.html;

    ssl_certificate /etc/ssl/certs/nginx-selfsigned.crt;
    ssl_certificate_key /etc/ssl/private/nginx-selfsigned.key;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;
    location / {
            proxy_pass http://backend;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
    }
}
```

Directives Explained:

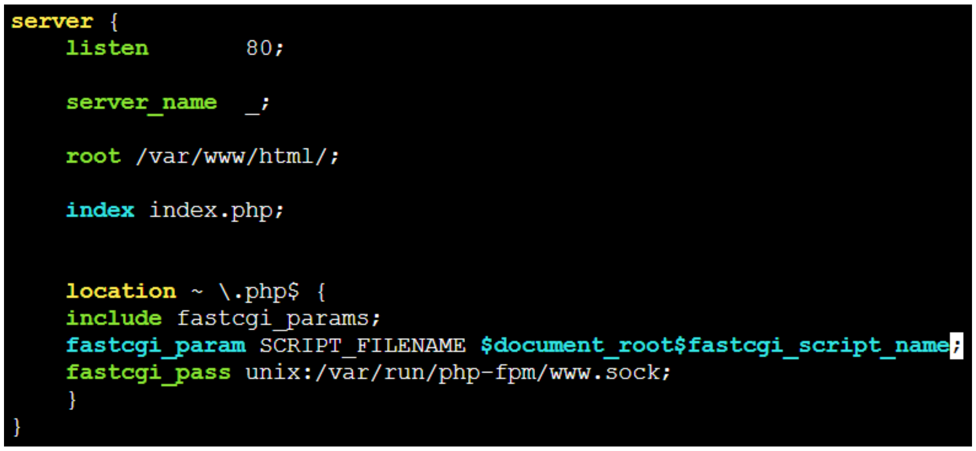
- Proxy_http_version 1.1: WS is not supported on HTTP v1.0 so we upgrade to 1.1
- Proxy_set_header Upgrade $http_upgrade: Passes the client's Upgrade: websocket header to the backend.
- Proxy_set_header Connection "upgrade": Tells Nginx to keep the connection open for upgrade.

Task2.7: Integrate PHP using PHP-FPM and fastcgi_pass

Nginx does not run live websites (e.g. PHP, Node.js) code itself. Instead, it forwards PHP code stored on the disk to the PHP-FPM over a Unix socket where it is executed and turned into plain HTML and returned to Nginx. Nginx then forwards this to the client. PHP-FPM runs as a separate process and can be run on worker nodes as well. To configure this feature, a few packages must be installed first:

- *sudo apt update*
- *sudo apt install -y nginx php-fpm php-mysql*
- *# php-mysql for MySQL/MariaDB support; add other php-extensions as needed*

Then we need to edit the nginx.conf to pass the PHP-FPM.

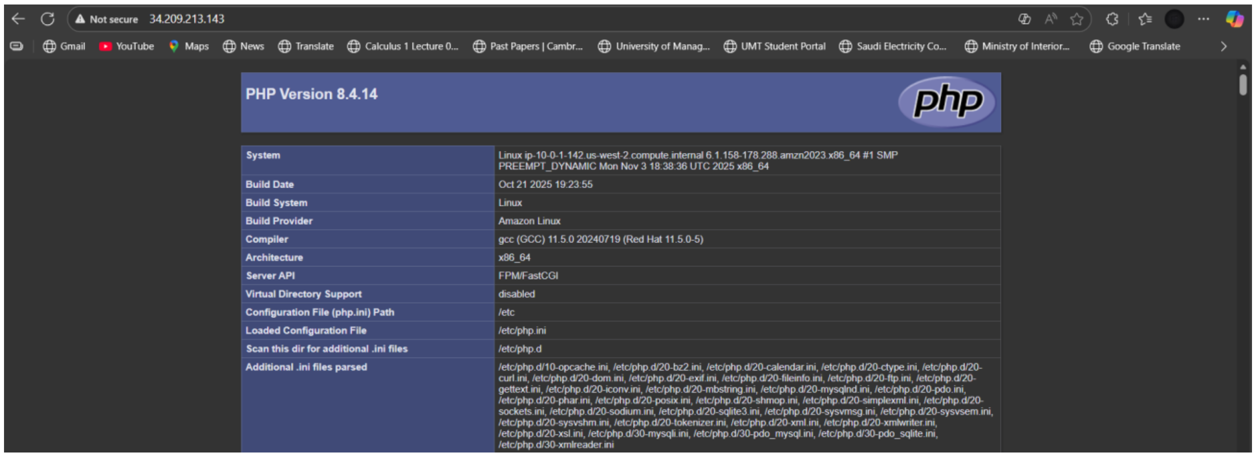


```
server {
    listen       80;

    server_name  _;

    root /var/www/html/;

    index index.php;


    location ~ \.php$ {
    include fastcgi_params;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    fastcgi_pass unix:/var/run/php-fpm/www.sock;
    }
}
```

Configuration Explained:

- Index index.php: Mention the name of the PHP file.
- Include fastcgi_params: This includes the config and parameters for PHP-FPM.
- Fastcgi_pass: This is the Unix socket location where the index.php will be passed.

Now when we access the IP of the instance, we can see the PHP Page loads:

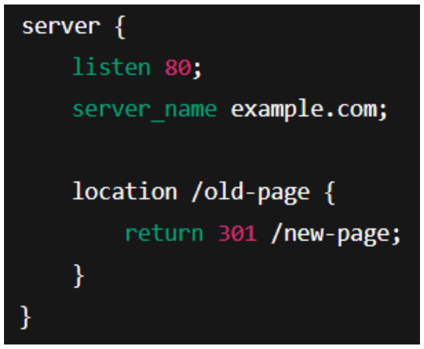


PHP Version 8.4.14

System	Linux ip-10-0-1-142.us-west-2.compute.internal 6.1.158-178.288.amzn2023.x86_64 #1 SMP PREEMPT_DYNAMIC Mon Nov 3 18:38:36 UTC 2025 x86_64
Build Date	Oct 21 2025 19:23:55
Build System	Linux
Build Provider	Amazon Linux
Compiler	gcc (GCC) 11.5.0 20240719 (Red Hat 11.5.0-5)
Architecture	x86_64
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc
Loaded Configuration File	/etc/php.ini
Scan this dir for additional .ini files	/etc/php.d
Additional .ini files parsed	/etc/php.d/10-opcache.ini, /etc/php.d/20-bz2.ini, /etc/php.d/20-calendar.ini, /etc/php.d/20-ctype.ini, /etc/php.d/20-curl.ini, /etc/php.d/20-dom.ini, /etc/php.d/20-exif.ini, /etc/php.d/20-fileinfo.ini, /etc/php.d/20-ftp.ini, /etc/php.d/20-gettext.ini, /etc/php.d/20-iconv.ini, /etc/php.d/20-mbstring.ini, /etc/php.d/20-mysqlnd.ini, /etc/php.d/20-pdo.ini, /etc/php.d/20-phar.ini, /etc/php.d/20-posix.ini, /etc/php.d/20-shmop.ini, /etc/php.d/20-simplexml.ini, /etc/php.d/20-sockets.ini, /etc/php.d/20-sodium.ini, /etc/php.d/20-sqlite3.ini, /etc/php.d/20-sysvmsg.ini, /etc/php.d/20-sysvsem.ini, /etc/php.d/20-sysvshm.ini, /etc/php.d/20-tokenizer.ini, /etc/php.d/20-xml.ini, /etc/php.d/20-xmlwriter.ini, /etc/php.d/20-xsl.ini, /etc/php.d/30-mysqli.ini, /etc/php.d/30-pdo_mysql.ini, /etc/php.d/30-pdo_sqlite.ini, /etc/php.d/30-xmlreader.ini

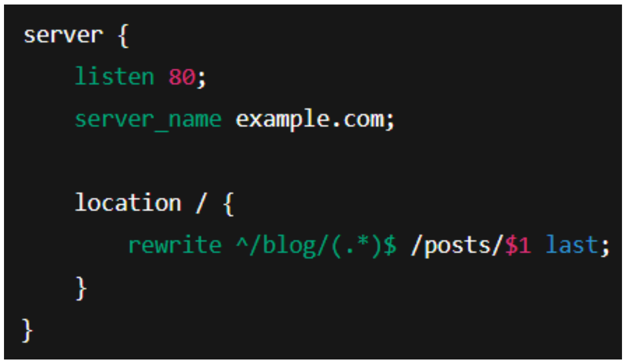
Task2.8: Set up URL rewriting and redirects

There are two types of redirects, permanent redirect (301) and temporary redirect (302). When the user is redirected, the browser is informed and the URL changes. We can setup redicts in nginx with the following configuration:



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

    location /old-page {
        return 301 /new-page;
    }
}
```

Rewrites modify the URL requested without updating it on the client's browser. We use the rewrite directive to rewrite the URL. Blow is an example:



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

    location / {
        rewrite ^/blog/(.*)$ /posts/$1 last;
    }
}
```

When the user requests for blog/article, they are served with /post/article without updating the URL in their browser.

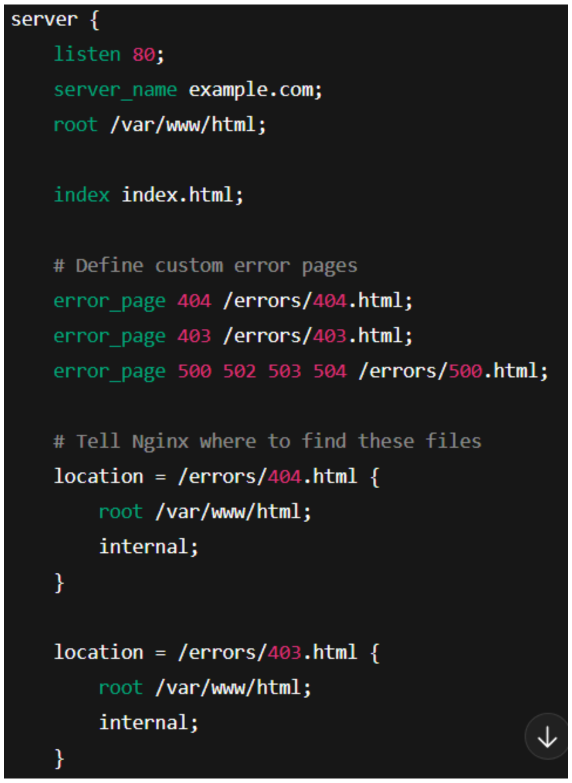
Task2.9: Add security headers and basic authentication

Security headers are extra HTTPS fields that tell the client's browser how to handle the website safely to prevent attacks such as XSS, Clickjacking, MIME-type sniffing, etc. Here are a few common security headers:

Header	Purpose	Example
Strict-Transport-Security (HSTS)	Forces HTTPS for future visits	Strict-Transport-Security: max-age=31536000; includeSubDomains
X-Content-Type-Options	Prevents browsers from guessing file types (MIME sniffing)	X-Content-Type-Options: nosniff
X-Frame-Options	Prevents your site from being loaded in iframes (defends clickjacking)	X-Frame-Options: SAMEORIGIN
X-XSS-Protection	Enables browser's XSS filter	X-XSS-Protection: 1; mode=block
Referrer-Policy	Controls how much referrer data is sent in links	Referrer-Policy: no-referrer-when-downgrade
Content-Security-Policy (CSP)	Controls what resources can load (prevents script injection)	Content-Security-Policy: default-src 'self'
Permissions-Policy	Restricts browser features like camera/mic/geolocation	Permissions-Policy: geolocation=(), camera=()

Task2.10: Create custom error pages (e.g., 404, 403)

By default, we have simple error pages that pop up on the client's browser when needed, but we can also create our own error pages and define them in our nginx.conf file to be displayed instead of the default error pages. Below is the configuration for custom error pages:

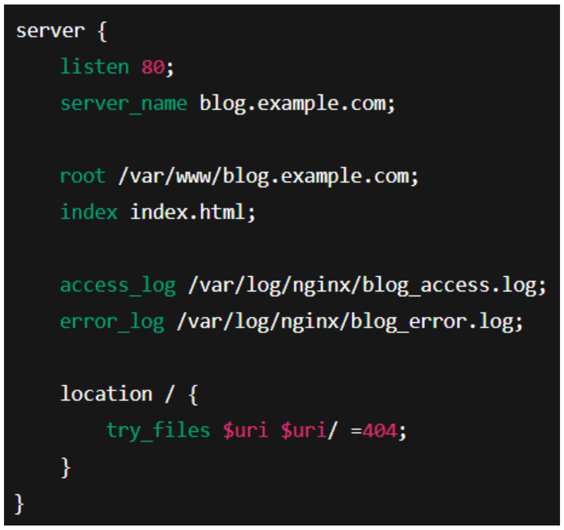


```
server {
    listen 80;
    server_name example.com;
    root /var/www/html;

    index index.html;

    # Define custom error pages
    error_page 404 /errors/404.html;
    error_page 403 /errors/403.html;
    error_page 500 502 503 504 /errors/500.html;

    # Tell Nginx where to find these files
    location = /errors/404.html {
        root /var/www/html;
        internal;
    }

    location = /errors/403.html {
        root /var/www/html;
        internal;
    }
```

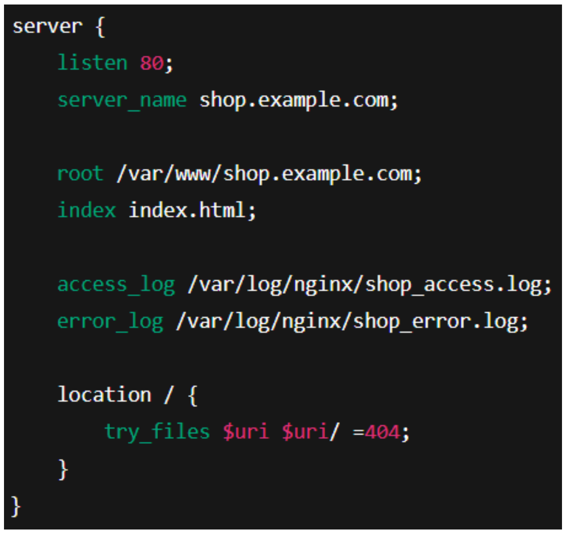
Here we used the error_page directive to define the error and then give the path variable to the error html file to display. The actual path is defined under the location block below.

Task2.11: Host multiple sites with subdomains and separate server blocks

Nginx can host multiple websites on a single IP address using domains and subdomains. They have separate root directories as well. In the nginx.conf file we define 2 servers with different names for different websites such as www.example1.com and www.example2.com. Here is a sample configuration.



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

    root /var/www/blog.example.com;
    index index.html;

    access_log /var/log/nginx/blog_access.log;
    error_log /var/log/nginx/blog_error.log;

    location / {
        try_files $uri $uri/ =404;
    }
}
```

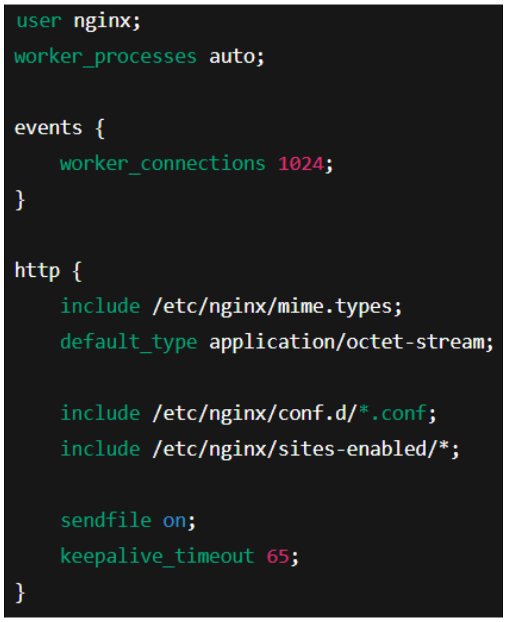


```
server {
    listen 80;
    server_name shop.example.com;

    root /var/www/shop.example.com;
    index index.html;

    access_log /var/log/nginx/shop_access.log;
    error_log /var/log/nginx/shop_error.log;

    location / {
        try_files $uri $uri/ =404;
    }
}
```

We can also create separate configuration files for each website and use the include directive inside the main nginx.conf file and that would work the same way as well.

Task2.12: Understand and modify nginx.conf using include directive

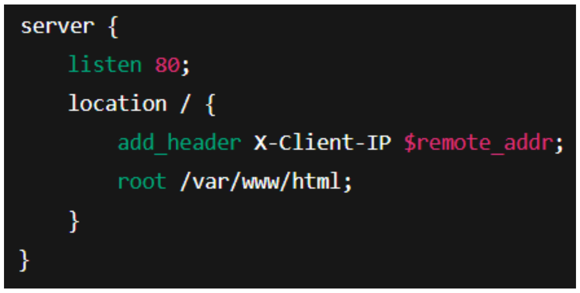
We can divide the configuration into multiple files to manageability and modularization by simply adding them into the main nginx.conf file using the include directive. When nginx starts it runs the nginx.conf file and loads all the other config files included as well.



```
user nginx;
worker_processes auto;

events {
    worker_connections 1024;
}

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;

    sendfile on;
    keepalive_timeout 65;
}
```

Task2.13: Use variables like $host, $remote_addr, $request_uri

Like any other programming language, nginx also supports variables in which we can store strings. Variables are useful for storing information such as remote address, host name, requested uri, etc.



```
server {
    listen 80;
    location / {
        add_header X-Client-IP $remote_addr;
        root /var/www/html;
    }
}
```

Task2.14: Implement rate limiting, connection limits, and buffering settings

Nginx provides features that help us protect our website from being overloaded. There are a few features that help us with this, such as rate limiting, connection limiting, and buffering.

- Limit_req: Rate limiting helps prevent abuse or overload by restricting how many requests per second each client (usually identified by IP) can make. Nginx stores a record of each client in memory using a zone (limit_req_zone), then counts their requests over time.

```
http {
    limit_req_zone $binary_remote_addr zone=one:10m rate=5r/s;
}
```

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

    location / {
        limit_req zone=one burst=10 nodelay;
        root /var/www/html;
    }
}
```

  In this example, we define one zone of size 10 mb with a limit rate of 5 requests per second.

- Limit_conn: This directive controls how many simultaneous TCP connections can be established with a server.

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

```
location /downloads/ {
    limit_conn addr 2;
    root /var/www/html;
}
```

Task2.15: Customize access and error logs

Nginx provides us with two types of logs. Access logs store the requests that were made to the webserver, and Error logs save the errors that occur during the operations. These logs are valuable for troubleshooting and for security analysis of the webserver. We can customize the template upon which the logs are structured and stored. Make use of the log_format directive to define the structure of the log entries.

```
log_format main '$remote_addr - $remote_user [$time_local] "$request" '
                '$status $body_bytes_sent "$http_referer" '
                '"$http_user_agent" "$http_x_forwarded_for"';
```

Logs storage can be defined by access log directive for access logs and error_log directive for error logs.

```
access_log /var/log/nginx/access.log;
```

```
error_log /var/log/nginx/error.log warn;
```