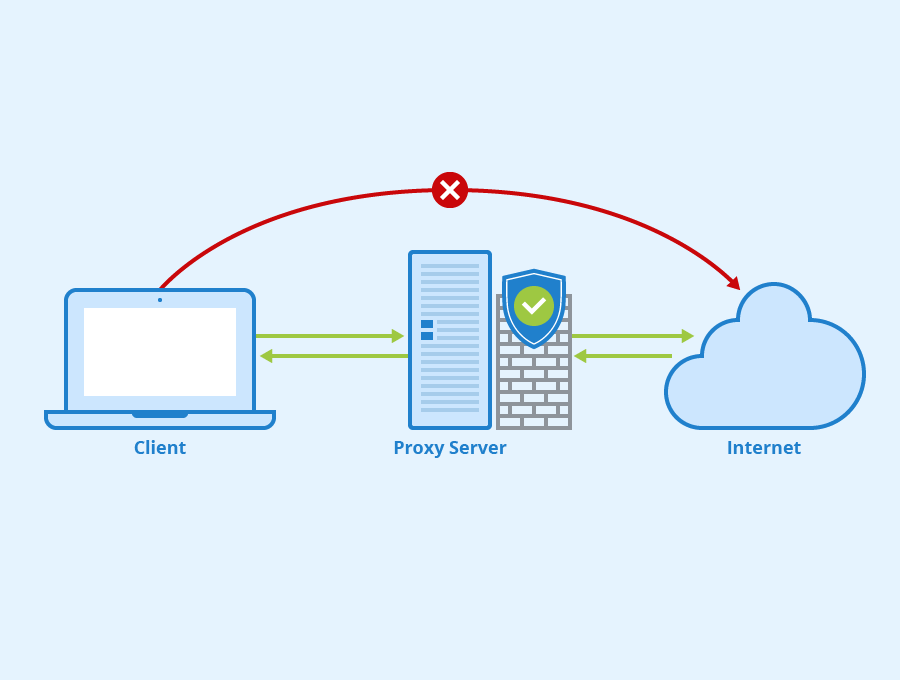
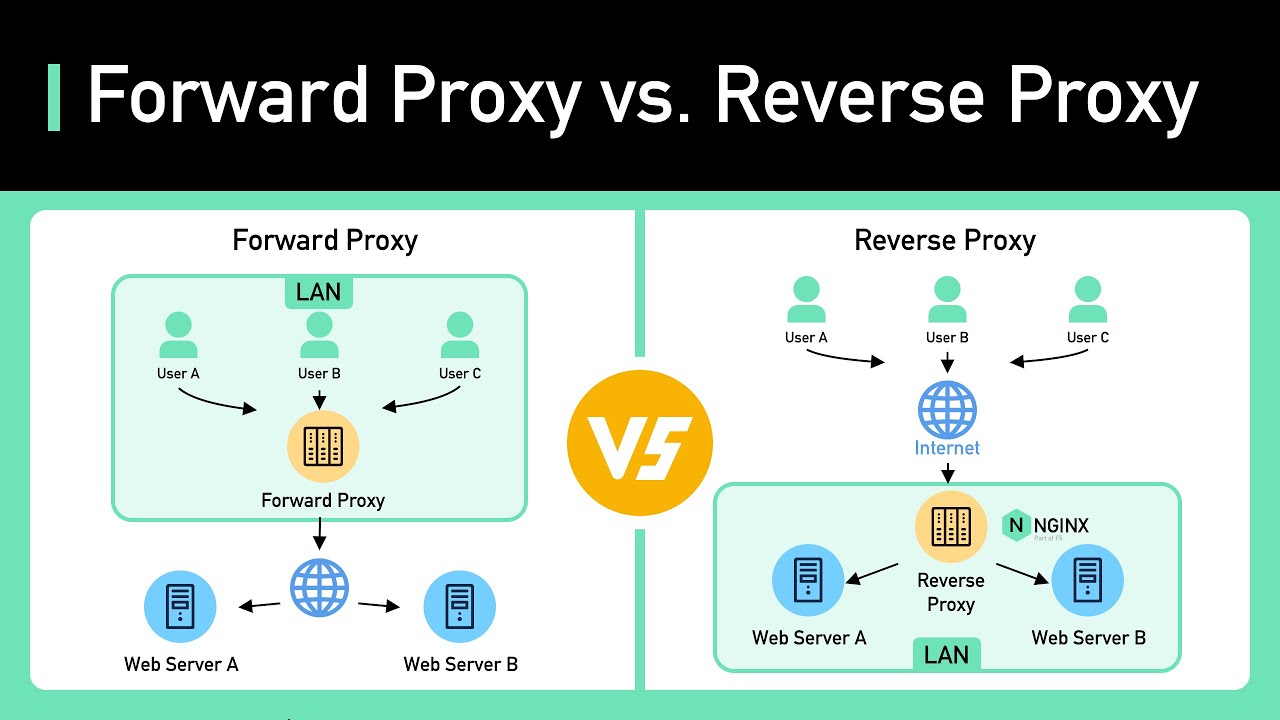
**NGINX Reverse Proxy Setup with Example Using HTTPD [Hindi]**

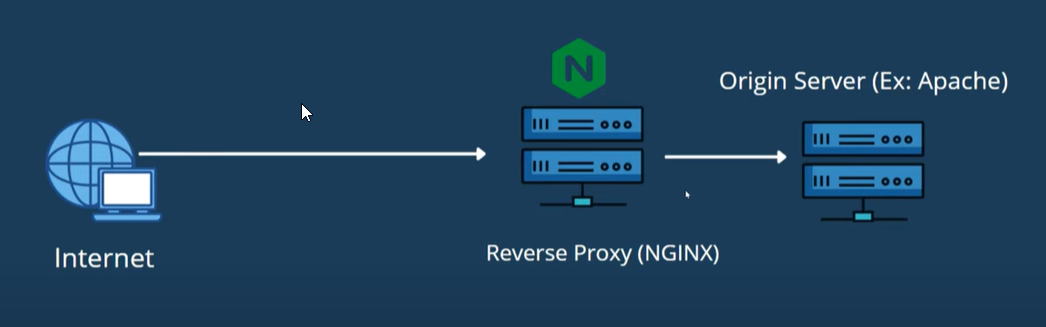
* In computer networking, a proxy server is a server application that acts as an intermediary between a client requesting a resource and the server providing that resource



* There are 2 types of proxy
  + Forward proxy – if proxy is present on the client end
  + Reverse proxy – if proxy is present on the server end



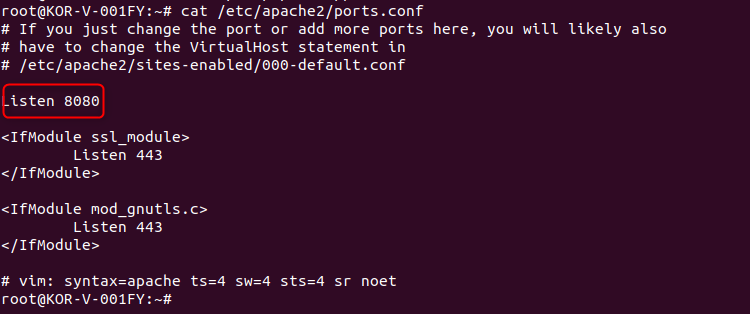
* When we use VPN, vpn is a proxy on the client end so it’s a forward proxy
* Reverse Proxy – a proxy service which takes a client request, passes it on to one or more servers
* Proxying is typically used to distribute the load among several servers, seamlessly show content from different websites or pass requests for processing to application servers over protocols other than HTTP
* Idea is to setup reverse proxy on the setup using nginx and the our server will be using apache for connecting to the web
* Meaning we have apache web server is present on our ubuntu machine, on this server our website is deployed now we will setup nginx as reverse proxy so that whenever we try to access our website using apache server, apache server will go through nginx web server
* Meaning when we try to access nginx wen server, we will get content of apache server(God knows kya bol rha hai yeh banda, dekhte hai end tak pata chalega toh update kardungi yeh lines)



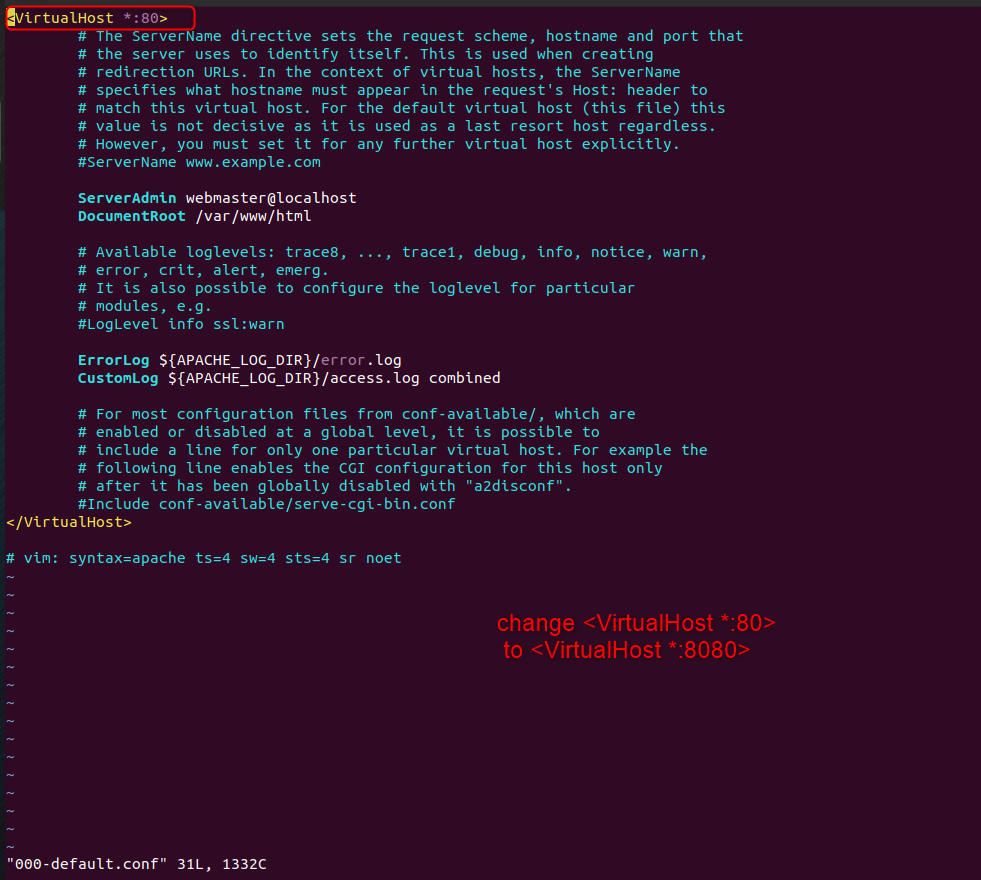
* For this we have to make sure both apache2 and nginx are running, but since both services are listening on port 80, we will have to change one service to a different port
* Decided to put apache on port 8080
* In the below file change listening port to 8080 - sudo nano /etc/apache2/ports.conf



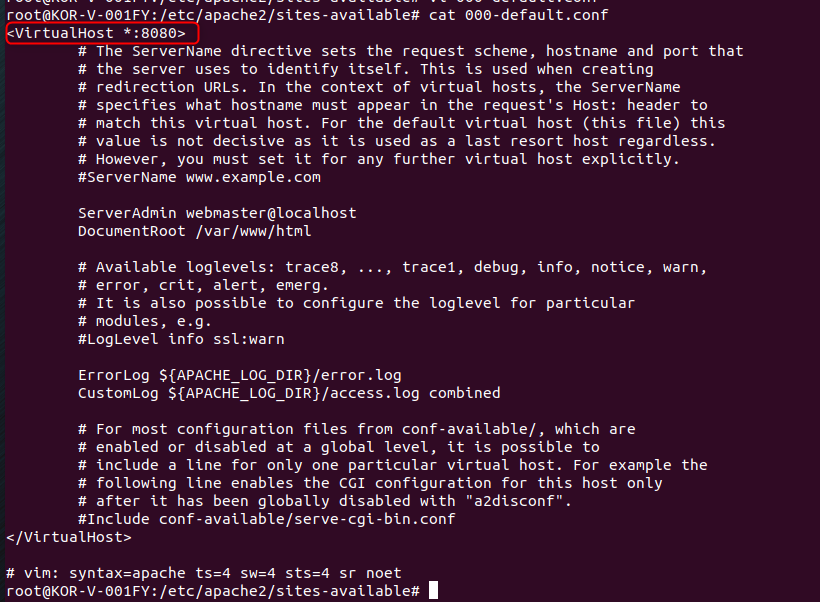
* The updated ports.conf looks like this



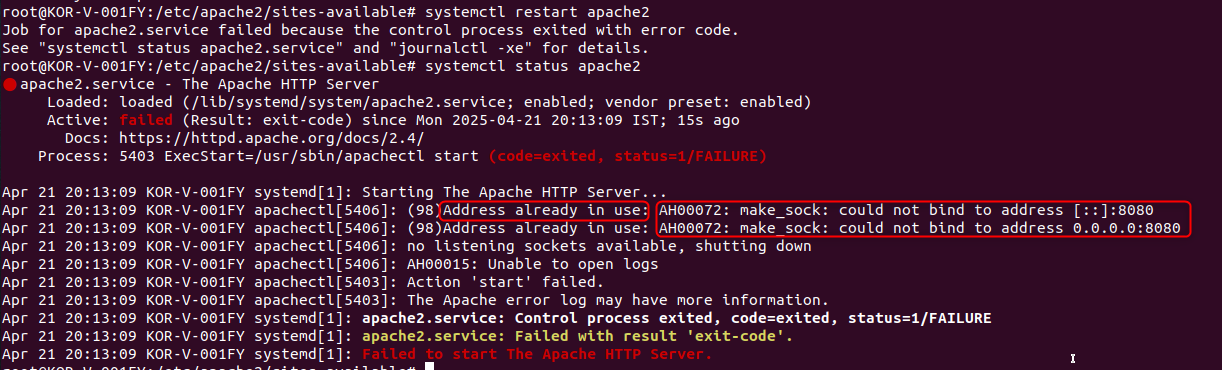
* Second thing is update the default virtual host - sudo nano /etc/apache2/sites-available/000-default.conf



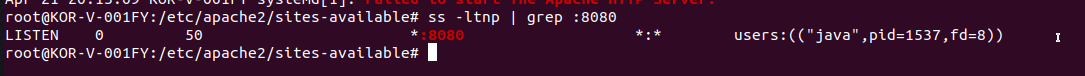
* Now the updated 000-default.conf looks like this –



* Last step is to just restart the service - sudo systemctl restart apache2
* Here I am again seeing error saying 8080 port is already being used



* Lets sirst find which service is using port 8080 using command - sudo ss -ltnp | grep :8080
* This port is being used by java, I don’t want to stop java service



* Ok lets try to find which ports are not in use using this simple bash script

# List every TCP port in 8000‑9000 that is NOT in use

comm -23 \

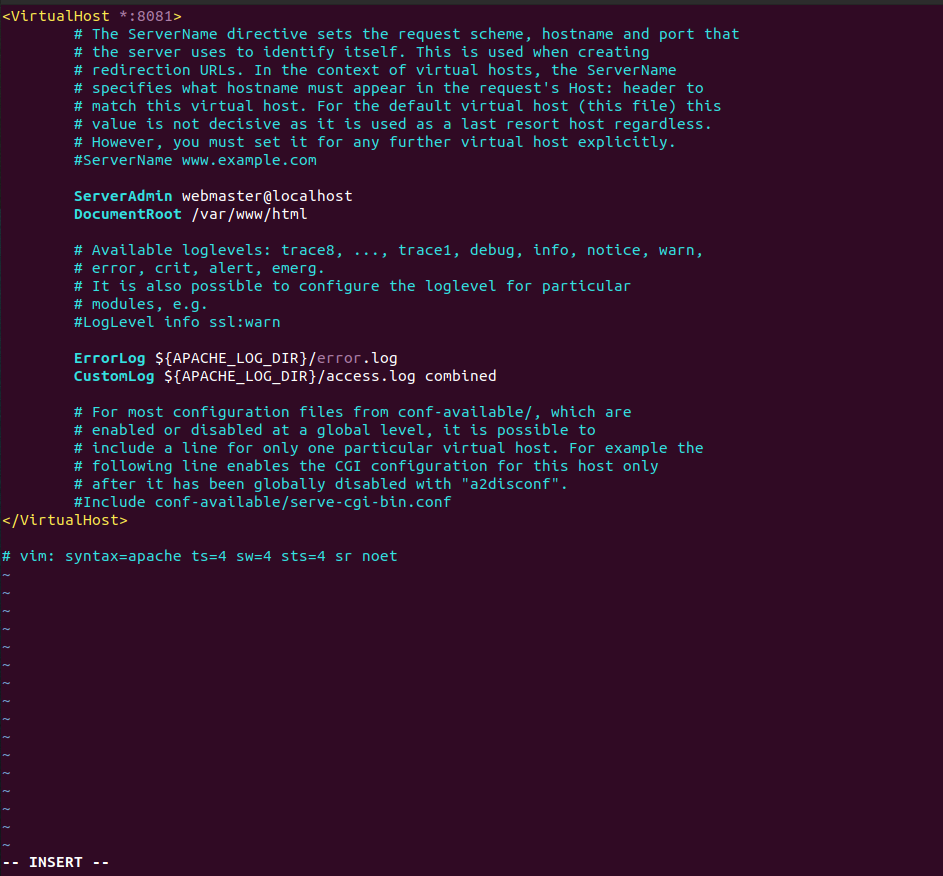
<(seq 8000 9000 | sort) \

<(sudo ss -tln | awk 'NR>1 {split($4,a,":"); print a[length(a)]}' | sort -n | uniq)

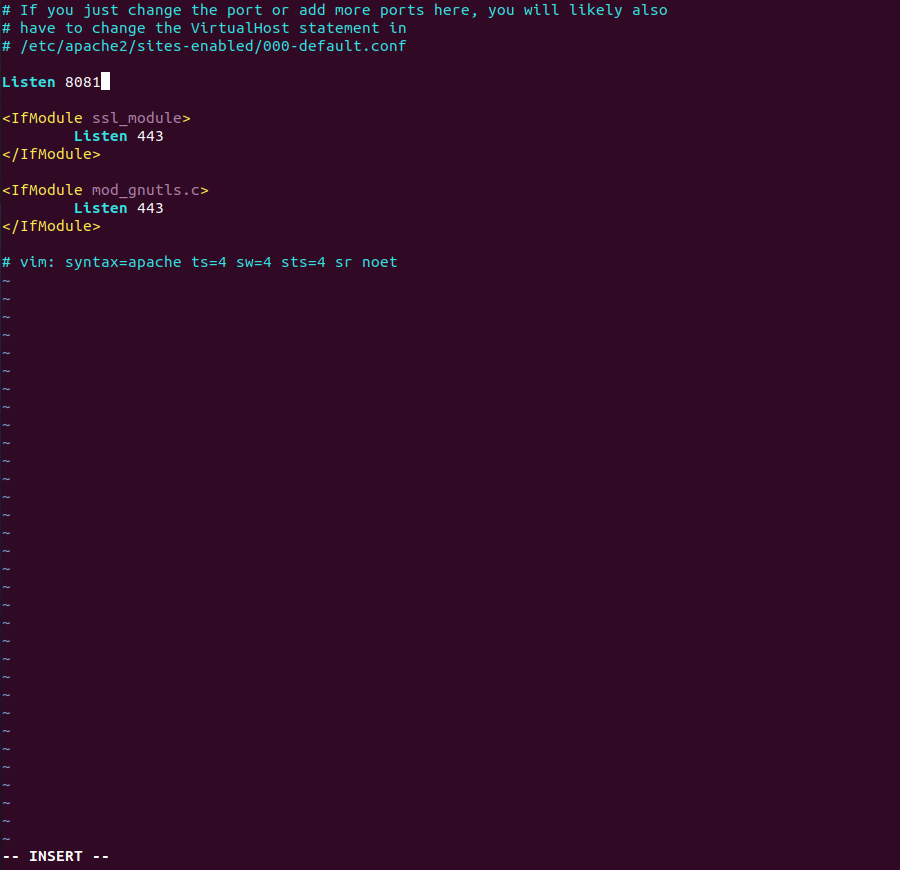
* 8081 port also seems to be free, I will verify it again using – sudo ss -ltnp | grep :8081



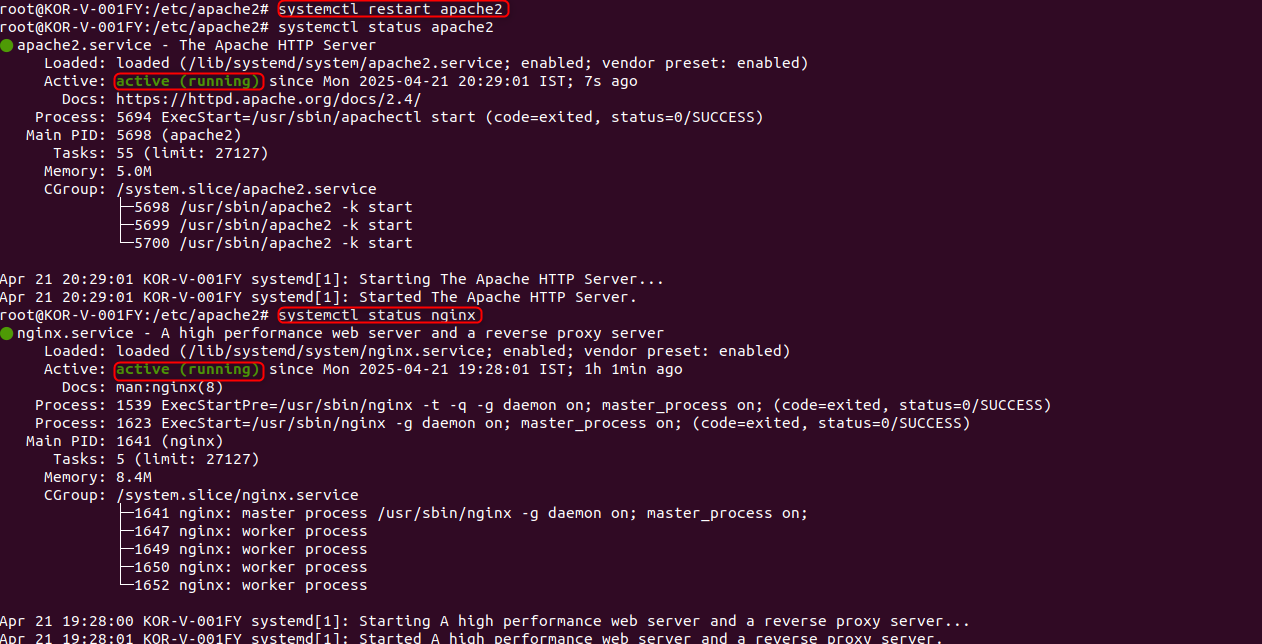
* No one is using 8081 port so we will again just change 8080 to 8081 in ports.conf and 000-default.conf
* In 000-default.conf



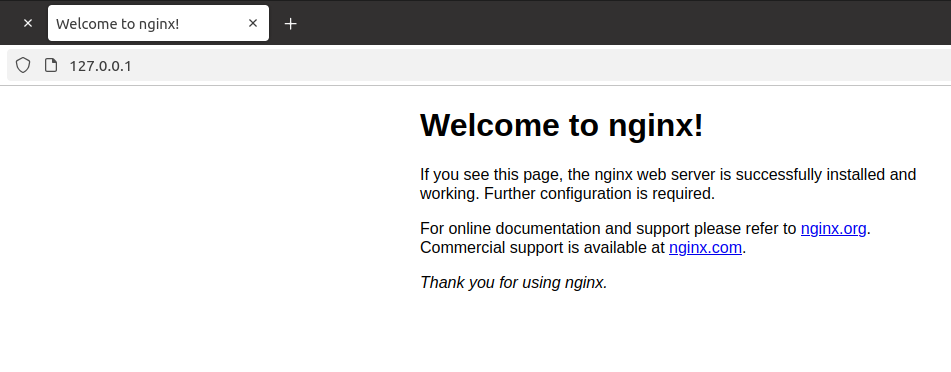
* In ports.conf



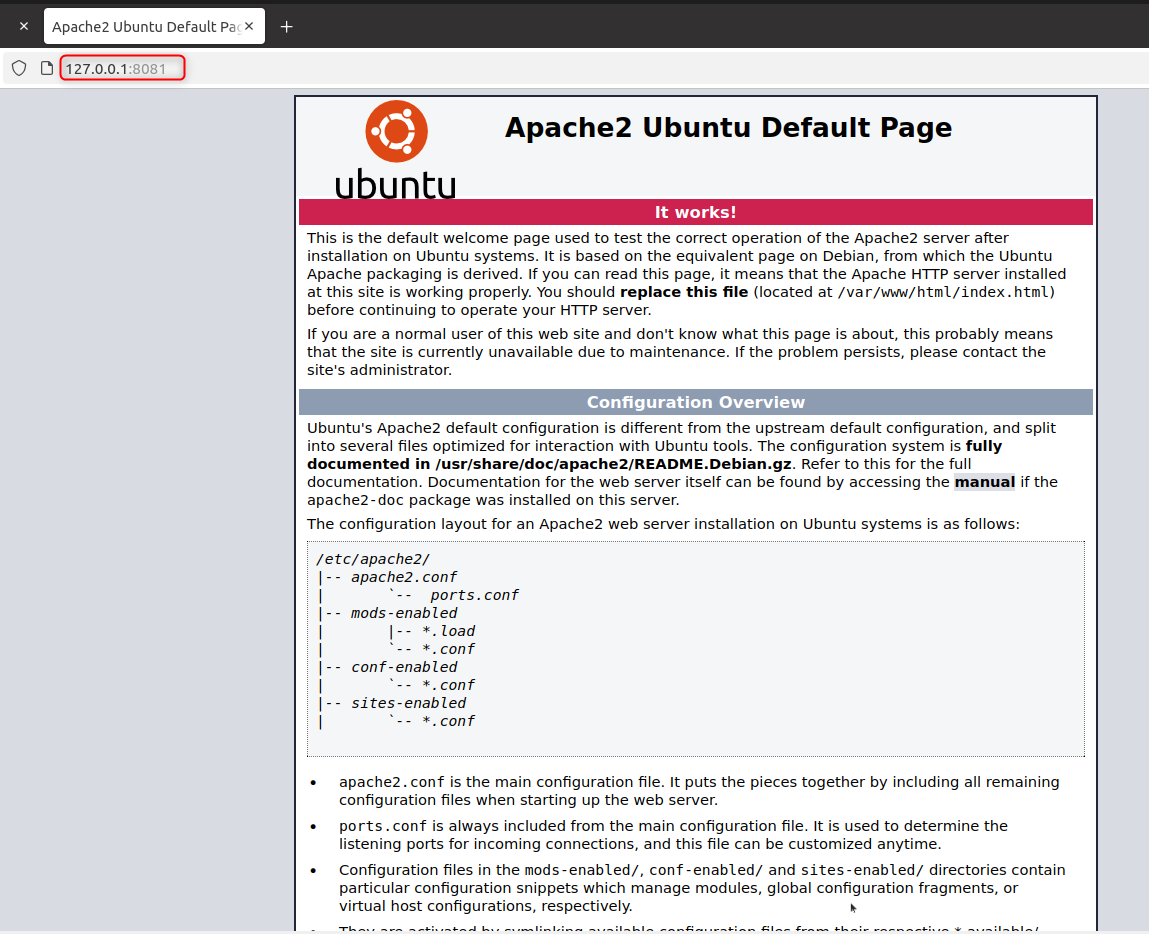
* Then restart the service again –
* Now its working , yayayaya, now we have apache2 and nginx both working



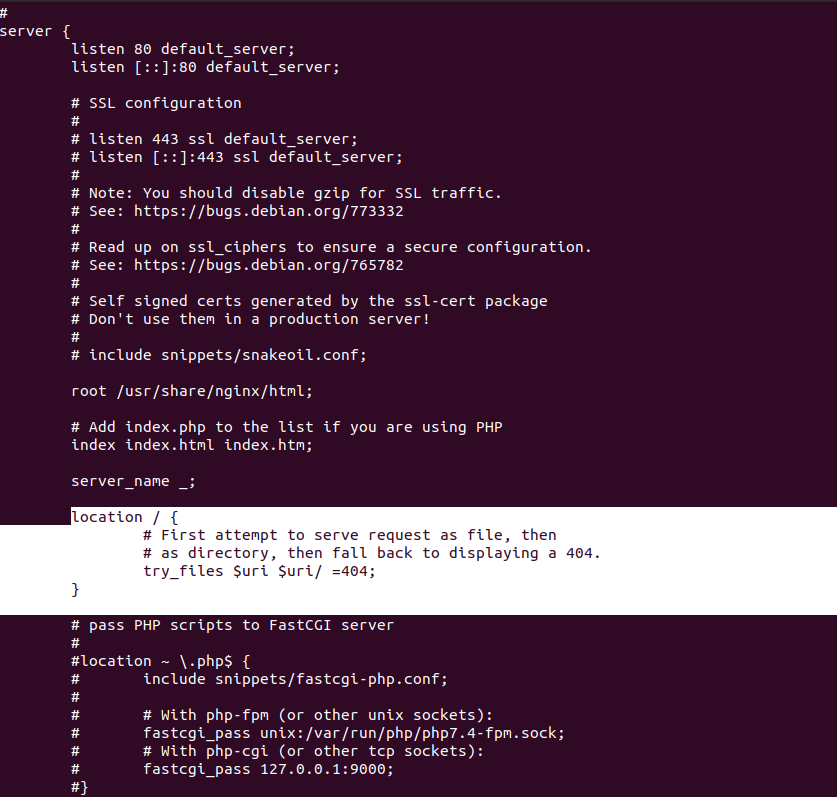
* Currently I have kept firewalld as active since in last video I was able to access the website, in case you are not try to stop the firewalld service
* Now in browser when I run – 127.0.0.1:80 I get the nginx website



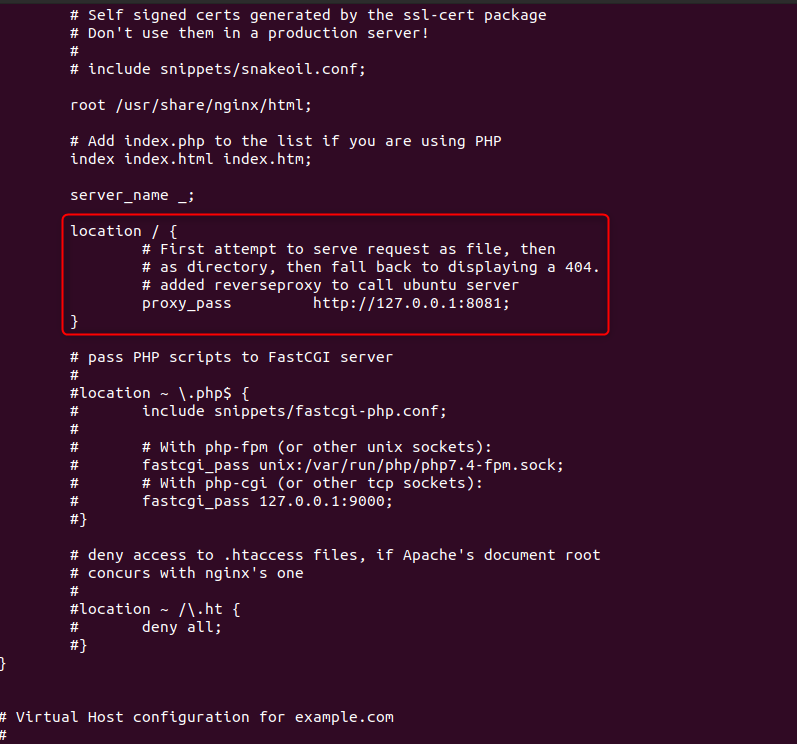
* And when I run 127.0.0.1:8081 I get the apache2 website



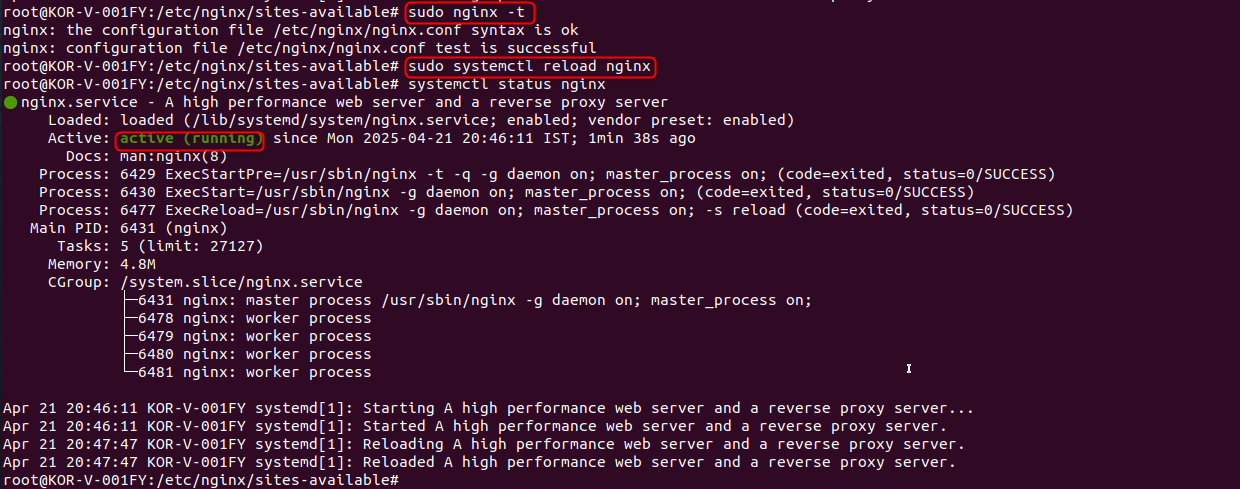
* Now for reverse proxy setup we have to do changes in file – sudo vi /etc/nginx/sites-available/default
* Earlier content in default –



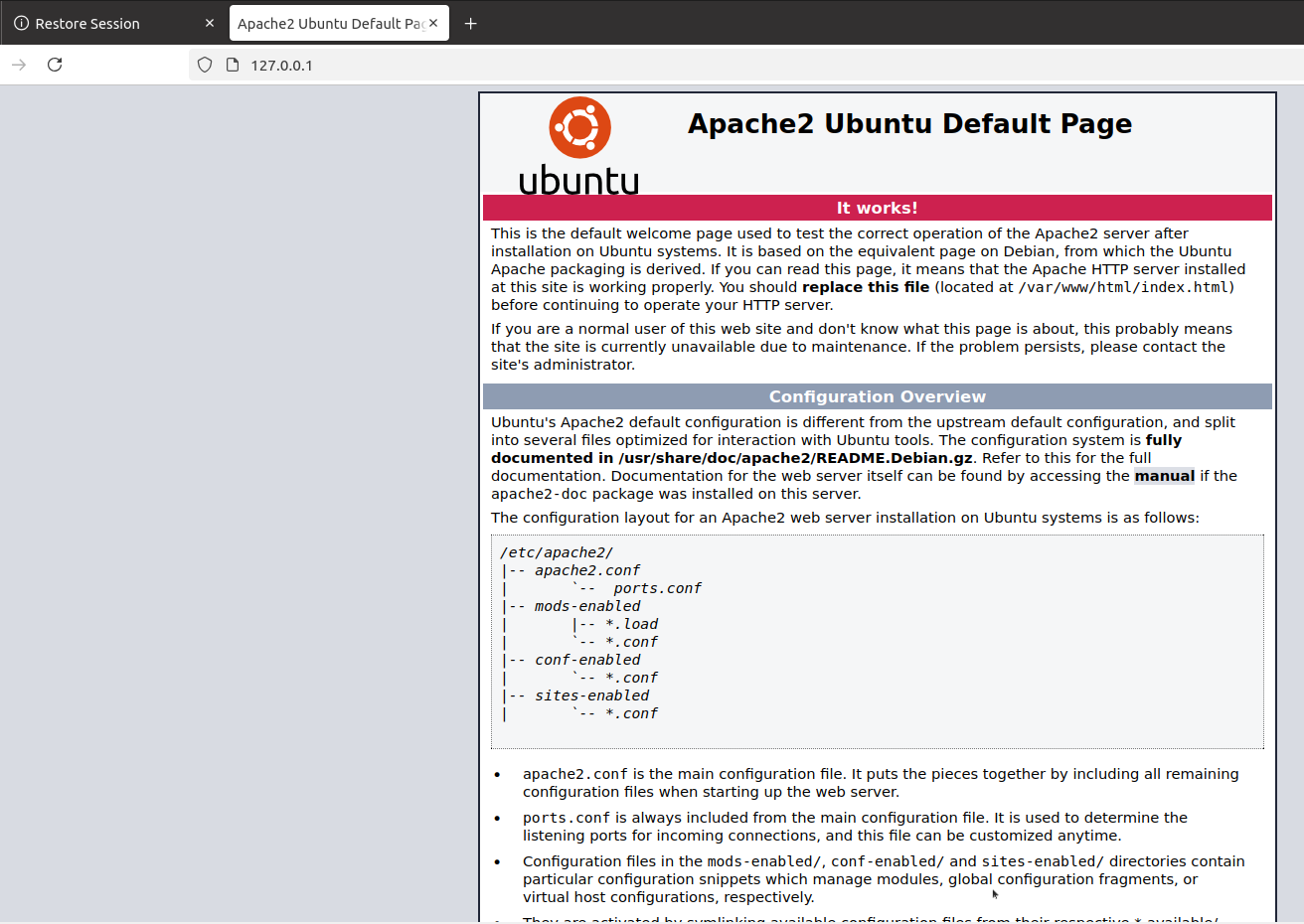
* Removing try\_files and replacing it with line - proxy\_pass <http://127.0.0.1:8081>;



* Now we will first test the nginx configuration - sudo nginx -t
* Next we will reload the nginx service, not restart we will reload using command – systemctl reload nginx



* Next now when we try to reload – 127.0.0.1:80 which was the nginx page, surprise, you see the apache page opening, means now nginx is acting as a reverse proxy to open apache2 page



* I got it in first go, but if it doesn’t work you can troubleshoot in logs - /var/log/nginx

**Reverse proxy in plain English**

Think of a reverse proxy (Nginx in your case) as **the receptionist at a big office**:

1. **You walk up to the front desk** → your browser connects to port 80 on the server (http://127.0.0.1).
2. **You say what you want** → the browser sends an HTTP request (“give me /”).
3. **The receptionist doesn’t actually have the document**.  
   *Instead* they know which employee does, so they quietly walk down the hall, ask that employee, get the paper, and bring it back to you.

The receptionist is Nginx, the employee is Apache that now lives on the private port 8081.  
To you, it looks as if the receptionist handed you the answer instantly—you never meet the employee or learn their room number.

**What happens under the hood**

1. **Browser ➜ Nginx**  
   *Destination*: server‑IP port 80  
   *Nginx’s job*: read the request headers and decide where to forward it.
2. **Nginx ➜ Apache (internal hop)**  
   *Nginx opens a new TCP connection* to 127.0.0.1:8081 and copies the original request across (proxy\_pass).  
   *Extra headers* (X‑Forwarded‑For, etc.) tell Apache who the real visitor is.
3. **Apache ➜ Nginx**  
   Apache generates the page and sends it back over that same internal connection.
4. **Nginx ➜ Browser**  
   Nginx streams Apache’s response back to the browser. The browser still thinks it has only ever talked to port 80.

**Why big sites use reverse proxies but you only type one URL**

* **Single public address** – [*www.google.com*](http://www.google.com) always points to port 443/80 of Google’s front‑end proxies.
* **Behind the scenes** those proxies dispatch the request to hundreds of internal services on many different private ports, servers, or even data‑centres.
* **DNS hides this**: you type one domain, the reverse‑proxy layer fans the traffic out where it needs to go.

**Recap**

*You* → **Nginx (port 80)** → **Apache (port 8081, private)** → **Nginx** → *You*

The reverse proxy is just an extra hop that lets one public port front many hidden services, handle SSL, load‑balance, cache, or add security—while visitors only ever see the single, friendly URL.