Simple and traditional CI/CD pipeline

https://medium.com/@saminchandeepa/auto-deploy-node-js-rest-api-on-aws-ec2-ci-cd-pipeline-using-github-actions-031be66824d1

First of all you may need proper virtual private server or shared server to act as a serving host to the intended application to be hosted. Other factor should be the github account.

In my case I am using azure Ubuntu 24.04.3 LTS virtual server which is coming as the part of student subscription in Azure

using ssh we can remotely sign in to the instance :

ssh <instance user name>@<public ip address>

Setting Up GitHub Actions

Navigate to your repository settings on GitHub and select Actions. Add a self-hosted runner settings -> actions -> runners -> new self-hosted runners

In GitHub Actions, runners are the machines that execute the jobs defined within a workflow. When a workflow is triggered by an event (such as a push to a branch or a pull request), the jobs within that workflow are sent to a runner for execution.

There are two primary types of runners:

GitHub-hosted runners
These are virtual machines provided and managed by GitHub. They come with pre-installed tools and the necessary runner application. GitHub handles the maintenance, upgrades, and security of these runners. They are available with various operating systems (Ubuntu Linux, Windows, macOS) and can be chosen based on the requirements of your workflow. Larger and more powerful runners are also available for specific needs, such as those with more CPU cores or GPU capabilities.

Self-hosted runners
These are machines that you provision and manage yourself. You install the GitHub Actions runner application on your own server, virtual machine, or container. Self-hosted runners offer greater control over the hardware, operating system, and installed software, which can be beneficial for specific use cases like requiring custom environments or access to internal resources. However, this also means you are responsible for their maintenance, security, and availability.

We are going to setting up a self-hosted runner in GitHub

select option “Linux based”
Then GitHub will be populated some of Linux command lines those would be downl0ad, install and configure runner Api on our VPS. Let’s see what the code lines and intension are.

# Create a whatever folder. Typically, this would be our production release going to resides
$ mkdir actions-runner && cd actions-runner

# Download the latest runner package
$ curl -o actions-runner-linux-x64-2.329.0.tar.gz -L https://github.com/actions/runner/releases/download/v2.329.0/actions-runner-linux-x64-2.329.0.tar.gz

#Optional: Validate the hash
$ echo "194f1e1e4bd02f80b7e9633fc546084d8d4e19f3928a324d512ea53430102e1d actions-runner-linux-x64-2.329.0.tar.gz" | shasum -a 256 -c

# Extract the installer
$ tar xzf ./actions-runner-linux-x64-2.329.0.tar.gz

Last step, run it!
$ ./run.sh
will be in place future

Create the runner and start the configuration experience
$ ./config.sh --url https://github.com/chamalkaMarasinghe/sample-ci-cd-pipeline-using-github-actions --token AZ7PF7LJ4EUR6YIN5KTXADDJEXJKA

Authentication
√ Connected to GitHub
Runner Registration
Enter the name of the runner group to add this runner to: [press Enter for Default]
Enter the name of runner: [press Enter for free-vm-student-profile] sample-ci-cd-pipeline-using-github-actions-runner
This runner will have the following labels: 'self-hosted', 'Linux', 'X64'
Enter any additional labels (ex. label-1,label-2): [press Enter to skip]
√ Runner successfully added
Runner settings
Enter name of work folder: [press Enter for _work]
√ Settings Saved.
chamalka@free-vm-student-profile:~/sample-ci-cd-pipeline-using-github-actions-dir$

But the best idea is copying the command lines at current moment from the GitHub and executing them because versions of the packages can be different in future.

Environment Setup for GitHub Actions

settings -> secrets and variables -> actions -> new repository secret

its recommended to add all of the properties as once into one key-value pair. Key is key, value is all properties

name : PORT

Secret : PORT="5005"

Setup Action Workflow

Actions -> Continuous integration -> Nodejs workflow -> configure

Initial workflow code boilerplate

```
# This workflow will do a clean installation of node dependencies, cache/restore them, build the source code and run tests across different versions of node

# For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-nodejs

name: Node.js CI

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:

  build:

    runs-on: ubuntu-latest

    strategy:
      matrix:
        node-version: [18.x, 20.x, 22.x]
        # See supported Node.js release schedule at https://nodejs.org/en/about/releases/

    steps:
    - uses: actions/checkout@v4
    - name: Use Node.js ${{ matrix.node-version }}
      uses: actions/setup-node@v4
      with:
        node-version: ${{ matrix.node-version }}
        cache: 'npm'
    - run: npm ci
    - run: npm run build --if-present
    - run: npm test
```

Revised script according to current codebase

```
# This workflow will do a clean installation of node dependencies, cache/restore them, build the source code and run tests across different versions of node

# For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-nodejs

name: Node.js CI

on:
  push:
    branches: [ "main" ]  trigger workflow only during push; not in pulls

jobs:
  build:                          run test have been removed since there is no test cases

    runs-on: self-hosted in self hosted runners

    strategy:
      matrix:
        node-version: [20.x] better to keep one version, otherwise workflow files will be created for each version
        # See supported Node.js release schedule at https://nodejs.org/en/about/releases/

    steps:
    - uses: actions/checkout@v4
    - name: Use Node.js ${{ matrix.node-version }}
      uses: actions/setup-node@v4
      with:
        node-version: ${{ matrix.node-version }}
        cache: 'npm'
    - run: npm ci
    - run: npm run build --if-present
    - run: |                        create .env file during bulidng the codebase
        touch .env
        echo "${{ secrets.ALL_SECRET_ENV }}" > .env
    - run: pm2 restart BackendAPI
           process manager 2 (pm2) to continuously running processes
```

Finally commit the changes to the branch and this will create a .yml workflow in the root directory of the source code

sample-ci-cd-pipeline-using-github-actions / .github / workflows / .yml file

Environment Setup in Ubuntu

Sudo apt update

Ensure Node.js and Nginx are installed on our Ubuntu instance.

curl -fsSL https://deb.nodesource.com/setup_lts.x | sudo -E bash -

sudo apt-get install -y nodejs nginx

then check node -v and npm -v

info about nginx

Web Server — NGINX can serve static content like HTML, CSS, and JavaScript files, as well as dynamic content generated by applications running on backend servers.

Reverse Proxy — NGINX can act as a reverse proxy server, sitting between clients and backend servers. It can distribute incoming requests to multiple backend servers based on various criteria such as load balancing algorithms, server health checks, or request routing rules. This helps improve the availability, scalability, and reliability of web applications.

Load Balancer — NGINX can function as a load balancer to distribute incoming traffic across multiple backend servers to ensure optimal resource utilization and prevent overload on any single server. It supports various load balancing algorithms and can be configured for fault tolerance and session persistence.

HTTP Cache — NGINX can cache frequently accessed content in memory or on disk to reduce latency and improve the performance of web applications. By caching static content or dynamically generated responses, NGINX can serve subsequent requests more quickly without involving the backend servers.

TLS/SSL Termination — NGINX can terminate TLS/SSL connections, offloading the encryption and decryption workload from backend servers.

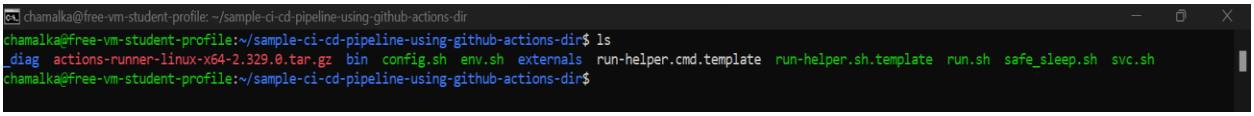
Reverse Proxy for Microservices — NGINX is commonly used as a reverse proxy for microservices architectures, where it can route requests to different microservices based on URL paths, headers, or other criteria.

Setting Up PM2

PM2 is a process manager for Node.js applications. Install and configure it to keep our application running in the background.

sudo npm i -g pm2 aaftertry pm2 command and test successfully installed

pm2 start server.js --name=apiServer



svc.sh is a shell script primarily used in the context of GitHub Actions self-hosted runners and Azure Pipelines agents on Linux systems. Its purpose is to manage the self-hosted runner or agent application as a system service, allowing it to run continuously in the background and automatically start on system boot.

sudo ./svc.sh install

output:
Creating launch runner in /etc/systemd/system/actions.runner.chamalkaMarasinghe-sample-ci-cd-pipeline-using-github-actions.sample-ci-cd-pipeline-using-github-actions-runner.service

Run as user: chamalka

Run as uid: 1000

gid: 1000

Created symlink /etc/systemd/system/multi-user.target.wants/actions.runner.chamalkaMarasinghe-sample-ci-cd-pipeline-using-github-actions.sample-ci-cd-pipeline-using-github-actions-runner.service → /etc/systemd/system/actions.runner.chamalkaMarasinghe-sample-ci-cd-pipeline-using-github-actions.sample-ci-cd-pipeline-using-github-actions-runner.service.

sudo ./svc.sh start

this will start the runners which were installed and configured previously

output:

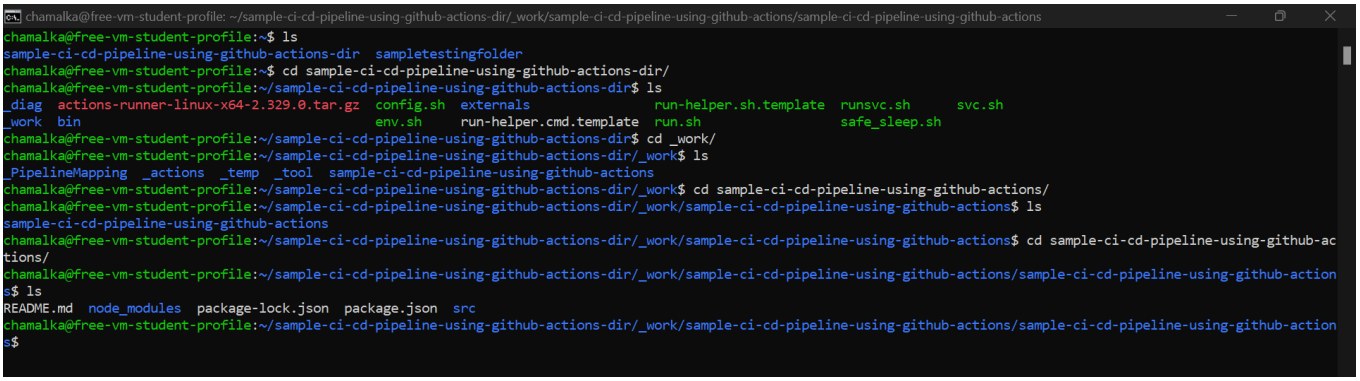
```
/etc/systemd/system/actions.runner.chamalkaMarasinghe-sample-ci-cd-pipeline-using-github-actions.sample-ci-cd-pipeline-using-github-actions-runner.service

● actions.runner.chamalkaMarasinghe-sample-ci-cd-pipeline-using-github-actions.sample-ci-cd-pipeline-using-github-actions-runner.service - GitHub Actions Runner (chamalkaMarasinghe-sample-ci-cd-pipeline-using-github-actions.sample-ci-cd-pipeline-using-github-actions-runner)

    Loaded: loaded (/etc/systemd/system/actions.runner.chamalkaMarasinghe-sample-ci-cd-pipeline-using-github-actions.sample-ci-cd-pipeline-using-github-actions-runner.service; enabled; preset: enabled)

    Active: active (running) since Tue 2025-11-25 17:06:30 UTC; 14ms ago

  Main PID: 5809 (runsvc.sh)

     Tasks: 2 (limit: 9446)

    Memory: 1.4M (peak: 1.4M)

       CPU: 8ms

    CGroup: /system.slice/actions.runner.chamalkaMarasinghe-sample-ci-cd-pipeline-using-github-actions.sample-ci-cd-pipeline-using-github-actions-runner.service

            ├─5809 /bin/bash /home/chamalka/sample-ci-cd-pipeline-using-github-actions-dir/runsvc.sh

            └─5812 ./externals/node20/bin/node ./bin/RunnerService.js


Nov 25 17:06:30 free-vm-student-profile systemd[1]: Started actions.runner.chamalkaMarasinghe-sample-ci-cd-pipeline-using-github-actions.sample-ci-cd-pipeline-...ons-runner).

Nov 25 17:06:30 free-vm-student-profile runsvc.sh[5809]: .path=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin

Hint: Some lines were ellipsized, use -l to show in full.
```

Whenever we have started the runner in our VPS it will also notify github that the runners are stared successfully, At this time if github have any queued actions those will be triggered immediately. By that we repo source code will be pushed into the VPS. It can be found in _work directory.

++
++

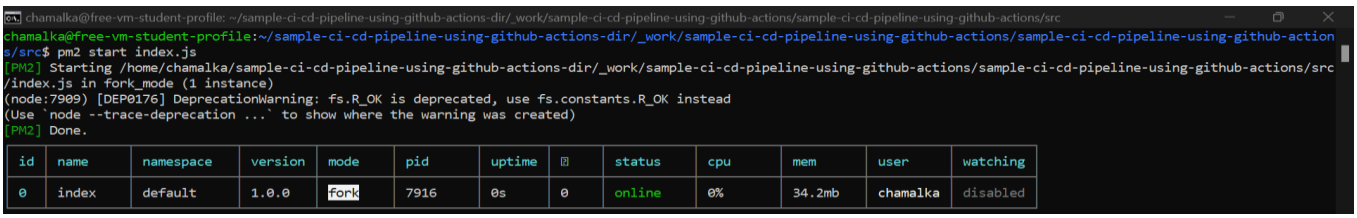
Fine; now we are going to do is go inside _work directory, find our actual source code and run that with pm2

navigate into our root level of our source code



Cd src/
going inside src folder, there is index.js file

pm2 start index.js or
Pm2 start index.js –name=<any name> or pm2 restart <app_id|app_name> --update-env
sudo pm2 restart index.js or
pm2 logs or pm2 ls



Now try to browse the public IP address of the VPS through web browser. Following output may bi visible in the web 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.

Setting Up Nginx Reverse Proxy

Configure Nginx to act as a reverse proxy for your Node.js application.

```
sudo nano /etc/nginx/sites-available/default
```

```
location /api/ {
  rewrite ^\/api\/(.*)$ /api/$1 break;
  proxy_pass http://localhost:5000;
  proxy_set_header Host $host;
  proxy_set_header X-Real-IP $remote_addr;
  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
}
```

rule for consider all requests after /api

Restart Nginx for changes to take effect,

```
sudo systemctl restart nginx
```

I faced an issue ??????????????????

I am developing a ci/cd pipeline using GitHub, git workflows, git actions I created a public repo pushed source code to it add GitHub secrets and variables -> new repository secrets. when I go to create new repository secrets it will shows up two inputs, name and secret. I added name as random string (lets say ALL_SECRET_ENV) and I added all the env key pair values to secret input.

ENVIRONMENT="DEV"
PORT="5005"
CLIENT="http://localhost:3005"

my GitHub source code does not include env files. I am trying to create env file and insert the repository secrets into this through the workflow. Then I created runners, self-hosted runners, configure runner in my dedicated VPS and build the connection. Then I wrote my workflow, hired node js action and so on. in VPS I installed pm2 to establish active processes and nginx to act as reverse proxy.

this is my workflow

This workflow will do a clean installation of node dependencies, cache/restore them, build the source code and run tests across different versions of node # For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-nodejs name: Node.js CI on: push: branches: ["main"] jobs: build: runs-on: self-hosted strategy: matrix: node-version: [20.x] # See supported Node.js release schedule at https://nodejs.org/en/about/releases/ steps: - uses: actions/checkout@v4 - name: Use Node.js ${{ matrix.node-version }} uses: actions/setup-node@v4 with: node-version: ${{ matrix.node-version }} cache: 'npm' - run: npm ci - run: npm run build --if-present

```
- run: |
    touch .env
    echo "${{ secrets.ALL_SECRET_ENV }}" > .env
- run: pm2 restart index
```

but iam facing an issue, when I checked my vps, there is created .env file. but it is totally empty file.

⚠ Do NOT use quotes " "

⚠ Ensure **new lines remain** (do NOT paste as a single line).

Then in workflow use **printf**, not echo, because GitHub escapes newlines:

```
- run: |
   printf "%s" "${{ secrets.ALL_SECRET_ENV }}" > .env
```

Because echo "${{ secrets.ALL_SECRET_ENV }}" collapses multi-line secrets depending on shell.

revised workflow file.

```
- run: npm ci
- run: npm run build --if-present


- name: Create .env file
  run: |
   printf "%s" "${{ secrets.ALL_SECRET_ENV }}" > .env
```

This will be solved the issue.

Nginx some basic configurations

==========

```
location /api {
  rewrite ^\/api\/(.*)$ /api/$1 break;
  proxy_pass http://localhost:5000;
  proxy_set_header Host $host;
  proxy_set_header X-Real-IP $remote_addr;
  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
}
```

Matches any request that starts with /api
Example:

- /api/users
- /api/login
- /api/v1/products

All routed into this block.

rewrite ^\/api\/(.*)$ /api/$1 break;
it rewrites /api/... to the exact same /api/....
Example:

- /api/users → rewrites to /api/users
  So this rule is redundant.

break means:

- Stop processing further rewrite rules and continue inside this same location block.

proxy_pass http://localhost:5000;

Forwards the request to your backend server running on port 5000.

But here's an important note. With this configuration:

NGINX will send the FULL path including /api to the backend.

User calls: /api/users
NGINX proxies: http://localhost:5000/api/users
So your backend should expect routes like:

/api/users
/api/login
/api/posts

Header forwarding

These headers pass original client information to your backend:

Header	Meaning
Host	Original domain requested by the client
X-Real-IP	Real client IP address
X-Forwarded-For	Proxy chain of client IPs

If our backend *does not* expect /api prefix

should change to:

```
location /api {
  proxy_pass http://localhost:5000/;
}
```

This removes /api from request before sending to backend.

Example:

User calls: /api/users

NGINX proxies: http://localhost:5000/users

Multiple Processes Within Same Server Scenario

Let's assume we have remote virtual private server. Let's say for example its public Ip address is - 100.200.300.400. We have planned to host several multiple applications within this server. let's assume within the VPS we have

nodejs API - running on port 5005
reactjs application - running on port 3005
springboot API - running on port 5008

in future maybe we are going to obtain three different domains and assigned to these applications. But for now, we need a way to access those applications independently using public Ip address of the VPS so let's see how this can be done this with nginx

apps like this:

Application	Port on VPS	Public Access (Nginx)
Node.js API	5005	http://100.200.300.400/node-api
React App	3005	http://100.200.300.400/react
Spring Boot API	5008	http://100.200.300.400/sb-api

sudo nano /etc/nginx/sites-available/default

```
server {
    listen 80;
    server_name _;

    # --------------------------
    # 1. Node.js API (Port 5005)
    # URL: http://<IP>/node-api
    # --------------------------
    location /node-api/ {
        proxy_pass http://localhost:5005/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    # --------------------------
    # 2. React App (Port 3005)
    # URL: http://<IP>/react
    # --------------------------
    location /react/ {
        proxy_pass http://localhost:3005/;
    }

    # --------------------------
    # 3. Spring Boot API (Port 5008)
    # URL: http://<IP>/sb-api
    # --------------------------
    location /sb-api/ {
```

```
server {
    listen 80;
    server_name _;

    ..... previous config .....

    # --------------------------
    # 3. Spring Boot API (Port 5008)
    # URL: http://<IP>/sb-api
    # --------------------------
    location /sb-api/ {
        proxy_pass http://localhost:5008/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

}
```

Notice the trailing slash: 

```
location /node-api/ {
   proxy_pass http://localhost:5005/;
}
```

our Request	Backend Request
/node-api/users	/users
/node-api/login	/login

backend should NOT have /node-api prefix.

sudo nginx -t
sudo systemctl reload nginx

Now you can access your apps like this
Node.js API
http://100.200.300.400/node-api
http://100.200.300.400/node-api/users
React App
http://100.200.300.400/react
Spring Boot API
http://100.200.300.400/sb-api

When obtain domains, create three server blocks:

```
server {
   listen 80;
   server_name api.myapp.com;
   proxy_pass http://localhost:5005;
}

server {
   listen 80;
   server_name panel.myapp.com;
   proxy_pass http://localhost:3005;
}

server {
   listen 80;
   server_name spring.myapp.com;
   proxy_pass http://localhost:5008;
}
```

Get free domain names and configure SSL configurations ======

In this section let's see how to obtain valid and free domain names for our applications and then how to configure secure SSL connections with https.

---

https://www.freehostia.com/

sign up for free now option
Plan – Chocolate hosting plan
Hosted domain – Enter your desired domain name and check the availability
Fill the account information of the second page
create a free cloud web hosting account
enter control panel password
complete the email verification

Menu -> My Domains -> Hosted Domains

https://cp.freehostia.com/domains/hosted/

---

https://www.duckdns.org/ → create sub domain as part of duckdns main domain → add ipv4 address of remote server

nslookup test-domain-chamalka-marasinghe.duckdns.org
ping test-domain-chamalka-marasinghe.duckdns.org

Workflow when directing to the application

Client (Browser / App)
↓
custom-domain.com
↓ (DNS → A record)
Ubuntu VPS (Public IP)
↓
Nginx (Reverse Proxy)
↓
Your Web API (localhost:PORT)

curl ifconfig.me – command displays the current vps public ip address

In nano editor, select text by mouse and then just right click to copy content.
Paste to nano editor, just right click

Now we have obtained free domain name from duckDNS platform and then linked remote server public ipv4 address to the created domain within duckDNS.

test-domain-chamalka-marasinghe.duckdns.org

Then we need to go inside the remote server. The remote server has already configured the nginx configuration file for testing purposes. But this time I'm going to create separate nginx config file for this domain.

sudo nano /etc/nginx/sites-available/test-domain-chamalka-marasinghe.duckdns.org.conf

adding following content to the new configuration file. This content is from the default nginx configuration file and same as to those content. Two entries for front end application and backend API same as previous.

```
server {
    listen 80;
    server_name test-domain-chamalka-marasinghe.duckdns.org;

    root /home/chamalka/apps/release-client;

    index index.html index.htm index.nginx-debian.html;

    location / {
        # First attempt to serve request as file, then
        # as directory, then fall back to displaying a 404.
        try_files $uri $uri/ /index.html;
    }

    location /testing-workflow/ {
        rewrite ^\/testing-workflow\/(.*)$ /api/$1 break;
        proxy_pass http://localhost:5005;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

}
```

Create a symbolic link to enable the configuration

sudo ln -s /etc/nginx/sites-available/test-domain-chamalka-marasinghe.duckdns.org.conf /etc/nginx/sites-enabled/

sudo nginx -t > test cone syntax correctness of configuration files

sudo systemctl reload nginx

ls -l /etc/nginx/sites-enabled/

But up to now this is not a secure connection since we have not configured ssl certifications. Therefore, we only can access this http channel.

http://test-domain-chamalka-marasinghe.duckdns.org/ > frontend app

http://test-domain-chamalka-marasinghe.duckdns.org/testing-workflow/ > node api

---

Up to Now we have not configured secure ssl connection, Now let's see how to obtain ssl certificate and enable secure connection.

use certbot to obtain a free ssl certificate from Let's Encrypt

The objective of Let's Encrypt and the ACME protocol is to make it possible to set up an HTTPS server and have it automatically obtain browser-trusted certificates without any human intervention. This is accomplished by running an ACME client on a web server. Let's Encrypt will be reducing the validity period of the certificates we issue. They currently issue certificates valid for 90 days, which will be cut in half to 45 days by 2028

sudo apt install certbot python3-certbot-nginx

run the following command to obtain and configure ssl
sudo certbot --nginx -d test-domain-chamalka-marasinghe.duckdns.org

After the successfully obtained the certification for the given domain , we can see following output.

```
Account registered.
Requesting a certificate for test-domain-chamalka-marasinghe.duckdns.org

Successfully received certificate.
Certificate is saved at: /etc/letsencrypt/live/test-domain-chamalka-marasinghe.duckdns.org/fullchain.pem
Key is saved at:         /etc/letsencrypt/live/test-domain-chamalka-marasinghe.duckdns.org/privkey.pem
This certificate expires on 2026-03-20.
These files will be updated when the certificate renews.
Certbot has set up a scheduled task to automatically renew this certificate in the background.

Deploying certificate
Successfully deployed certificate for test-domain-chamalka-marasinghe.duckdns.org to /etc/nginx/sites-enabled/test-domain-chamalka-marasinghe.duckdns.org.conf
Congratulations! You have successfully enabled HTTPS on https://test-domain-chamalka-marasinghe.duckdns.org
```

Certbot usually sets up automatic renewal. to test this run this

sudo certbot renew --dry-run

```
Saving debug log to /var/log/letsencrypt/letsencrypt.log

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Processing
/etc/letsencrypt/renewal/test-domain-chamalka-marasinghe.duckdns.org.conf
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Account registered.
Simulating renewal of an existing certificate for test-domain-chamalka-marasinghe.duckdns.org

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Congratulations, all simulated renewals succeeded:
  /etc/letsencrypt/live/test-domain-chamalka-marasinghe.duckdns.org/fullchain.pem (success)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Now check the two urls, those can be access with https secure channel. Web browsers are able identify those valid ceritifications.