DevOps Engineer Assessment for Laravel-Based Application

Repository:

- Fork the following repository: Laravel Todo List API.

Assignment Instructions:

Record the time of task completion of each section.

For local development

1. **Dockerize the Application**: Create a Docker Compose file to set up the application for local development. *Do not use pre-built solutions like Sail or Larasail.*
2. **Database and Caching Configuration**: Configure database and caching services within the Docker environment.

Let's Dockerize the application:
Explanations:

- Using php fpm 8.3 version as a pre build base image.
- Required dependencies has been download and setup inside container.
- After that, installing composer plugin of php.
- After that setting up working directory of docker container as /var/www/html.
- After setting up, copying all code inside container.
- Then after setting up dependencies of php inside container.
- Fully setting up php, let's create application key which is used for authentication the application.
- Now all done, let's expose the container port which helps to communicate with other services inside container.
- FInally to start php while container up, we can use command as 'php-fpm'.

```
ubuntu@ubuntu-jammy:~/laravel-todolist-api-main$ cat Dockerfile
# Use the official PHP 8.3 FPM image
FROM php:8.3-fpm

# Install system dependencies
RUN apt-get update && apt-get install -y \
    git \
    curl \
    libpng-dev \
    libonig-dev \
    libxml2-dev \
    zip \
    unzip \
    && docker-php-ext-install pdo_mysql mbstring exif pcntl bcmath gd

# Install Composer
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer

# Set working directory
WORKDIR /var/www/html

# Copy application files
COPY . .

# Install PHP dependencies
RUN composer install --no-dev --optimize-autoloader

# Generate application key
RUN php artisan key:generate

# Expose port 9000 for PHP-FPM
EXPOSE 9000

# Set appropriate file permissions
RUN chmod -R 775 storage && \
    chown -R www-data:www-data storage && \
    touch storage/logs/laravel.log && \
    chmod 664 storage/logs/laravel.log && \
    chown www-data:www-data storage/logs/laravel.log

# Run database migrations
RUN php artisan migrate --force

# Start PHP-FPM
CMD ["php-fpm"]

ubuntu@ubuntu-jammy:~/laravel-todolist-api-main$
```

Dockerizing application has been done.

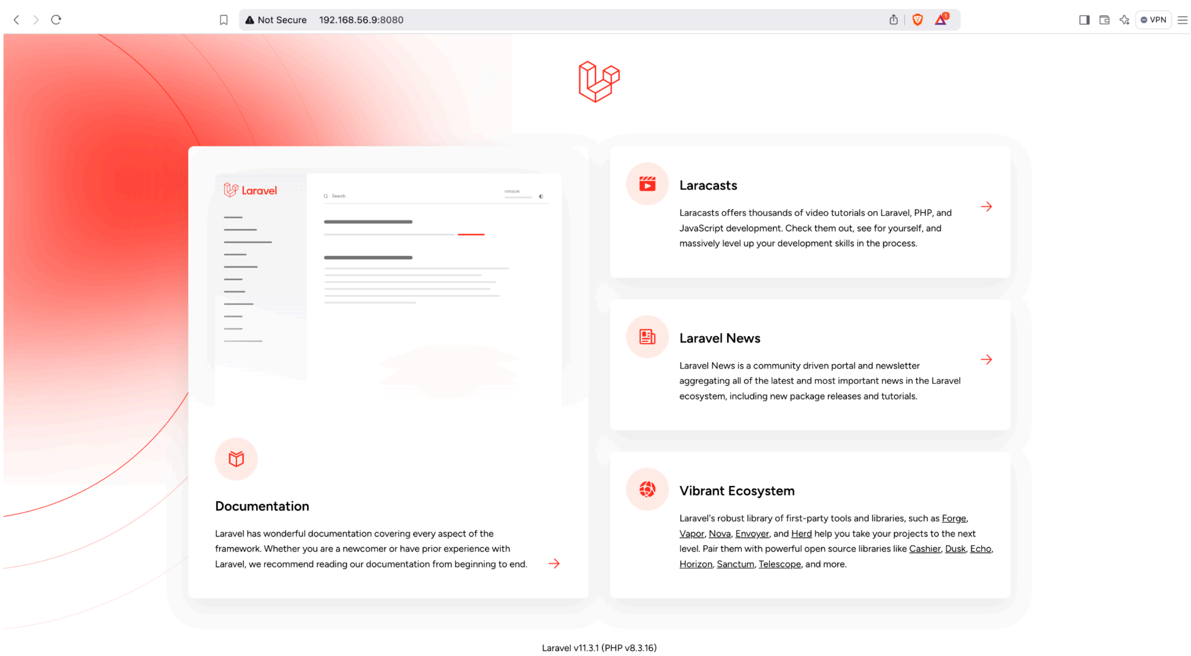
Let’s make docker-compose.yml file where I can use this image and pull other components for proper application setup:

```
ubuntu@ubuntu-jammy:~/laravel-todolist-api-main$ cat docker-compose.yml
version: '3.8'

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: laravel_app
    volumes:
      - .:/var/www/html
    working_dir: /var/www/html
    depends_on:
      - db
      - redis
    entrypoint: /bin/bash -c "composer install && php artisan key:generate

  web:
    image: nginx:alpine
    container_name: laravel_web
    ports:
      - "8080:80"
    volumes:
      - .:/var/www/html
      - ./docker/nginx.conf:/etc/nginx/conf.d/default.conf
    depends_on:
      - app

  db:
    image: mysql:8.0
    container_name: laravel_db
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: laravel
      MYSQL_USER: laravel
      MYSQL_PASSWORD: secret
    ports:
      - "3306:3306"
    volumes:
      - db_data:/var/lib/mysql
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
      interval: 5s
      timeout: 10s
      retries: 5
    command: --default-authentication-plugin=mysql_native_password

  redis:
    image: redis:alpine
    container_name: laravel_redis
    ports:
      - "6379:6379"

volumes:
  db_data:

ubuntu@ubuntu-jammy:~/laravel-todolist-api-main$
```

Now, dockers-compose.yml also ready with required components as webserver, database, redis for caching and application components .

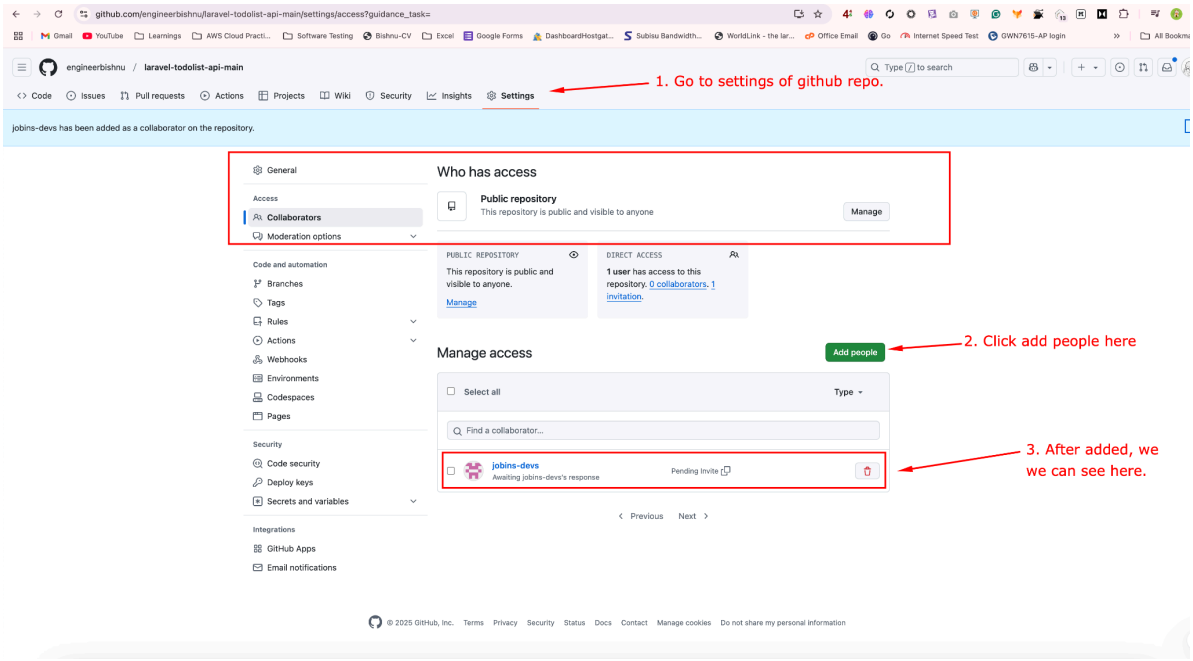
We can test locally whether it is working or not. I have tested, and it is working fine.

I have tested locally by using docker compose up –build, It build up and make containers up. Now I can browse the page with my ip:



Simple laravel browsing page has been open. Now we can perform our CRUD operationsn accordingly.

Let’s push this repository to my github account’s repo and add required user as collaborator:



After collaborator accept invitation, we can see the collaborator in manage access of repository settings.

Now, we can perform CRUD operation as required in README.md file in the repo:

Let’s Verify Application Functionality

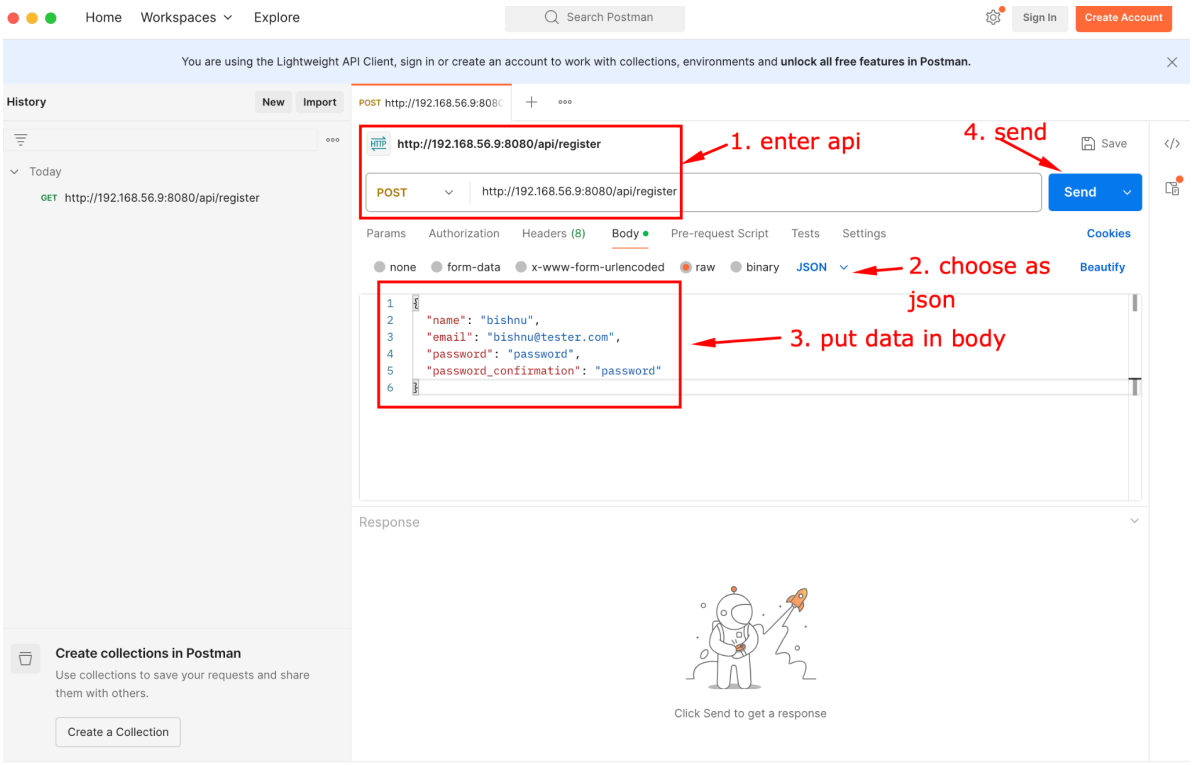
We can test the application using the following HTTP requests:

1. Register

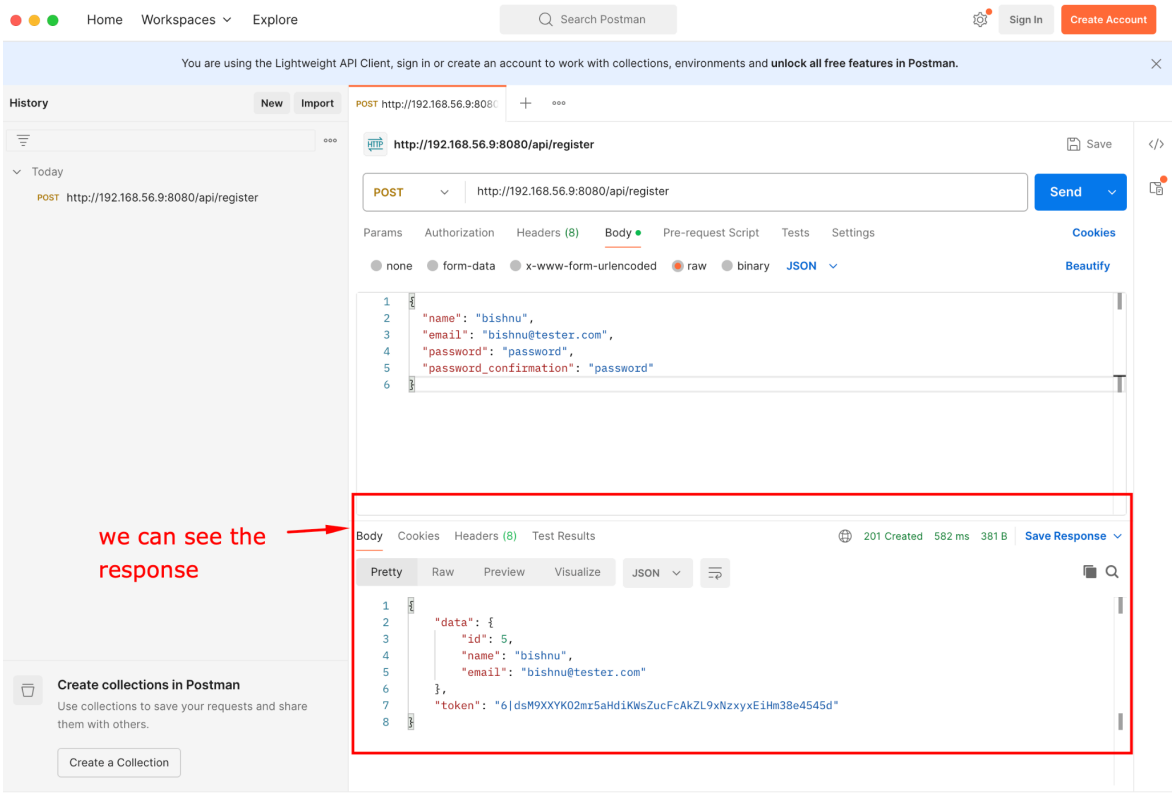
Request:

```
POST http://192.168.56.9:8080/api/register
Content-Type: application/json

{
  "name": "bishnu",
  "email": "bishnu@tester.com",
  "password": "password",
  "password_confirmation": "password"
}
```

Let’s do the test from Postman: Postman is a tool used for api testing purpose.



Put required api endpoint as shown above and put body data as json format shown above. And send it to the api:



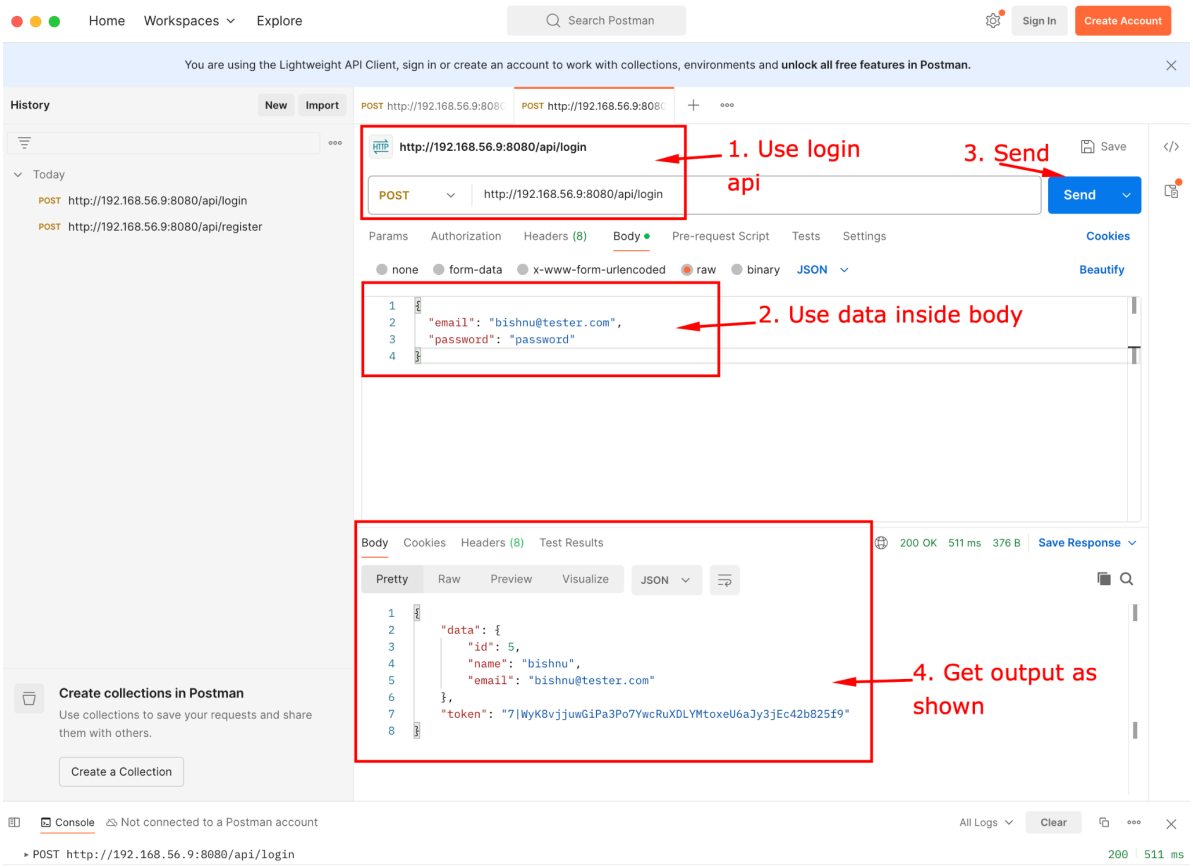
Above, we can see our user has been registered successfully.
Let's do further operations.

2. Login

Request:

```
POST http://192.168.56.9:8080/api/login
Content-Type: application/json

{
  "email": "bishnu@tester.com",
  "password": "password"
}
```

Inside postman:



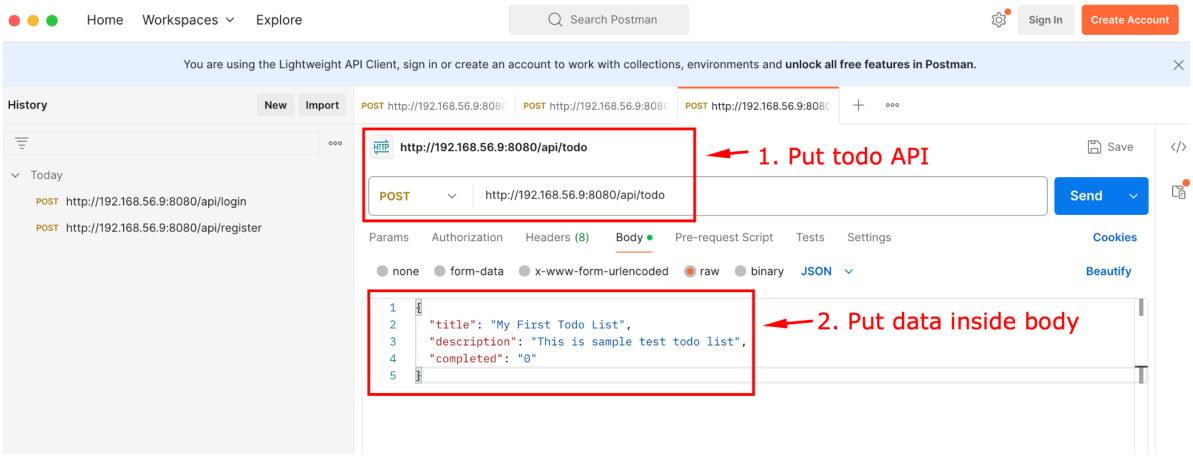
We can see, login operation also done.

3. Create To-Do Item

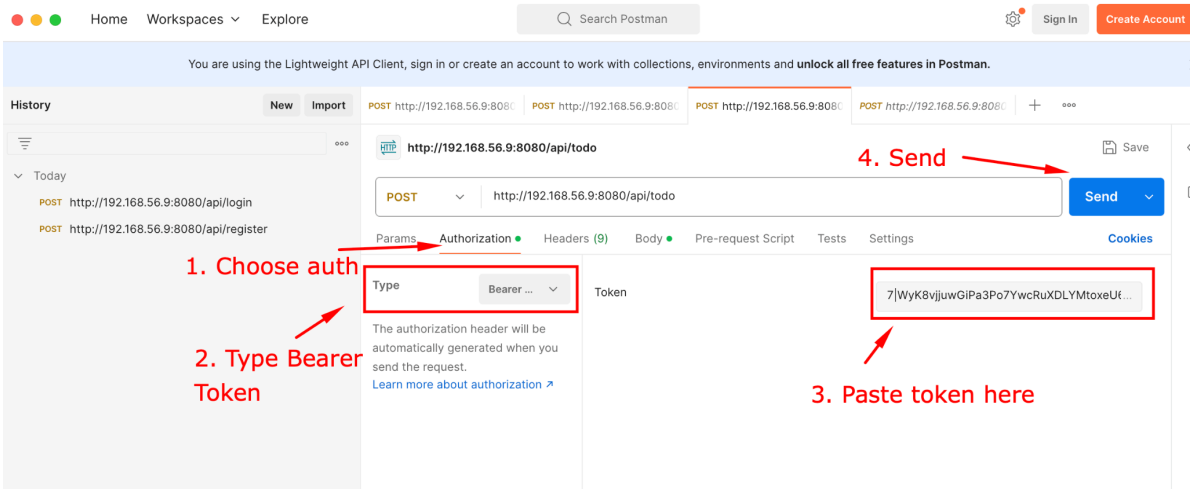
Request:

```
POST http://192.168.56.9:8080/api/todo

Authorization: Bearer YOUR_ACCESS_TOKEN
Content-Type: application/json

{
  "title": "My First Todo List",
  "description": "This is sample test todo list",
  "completed": "0"
}
```

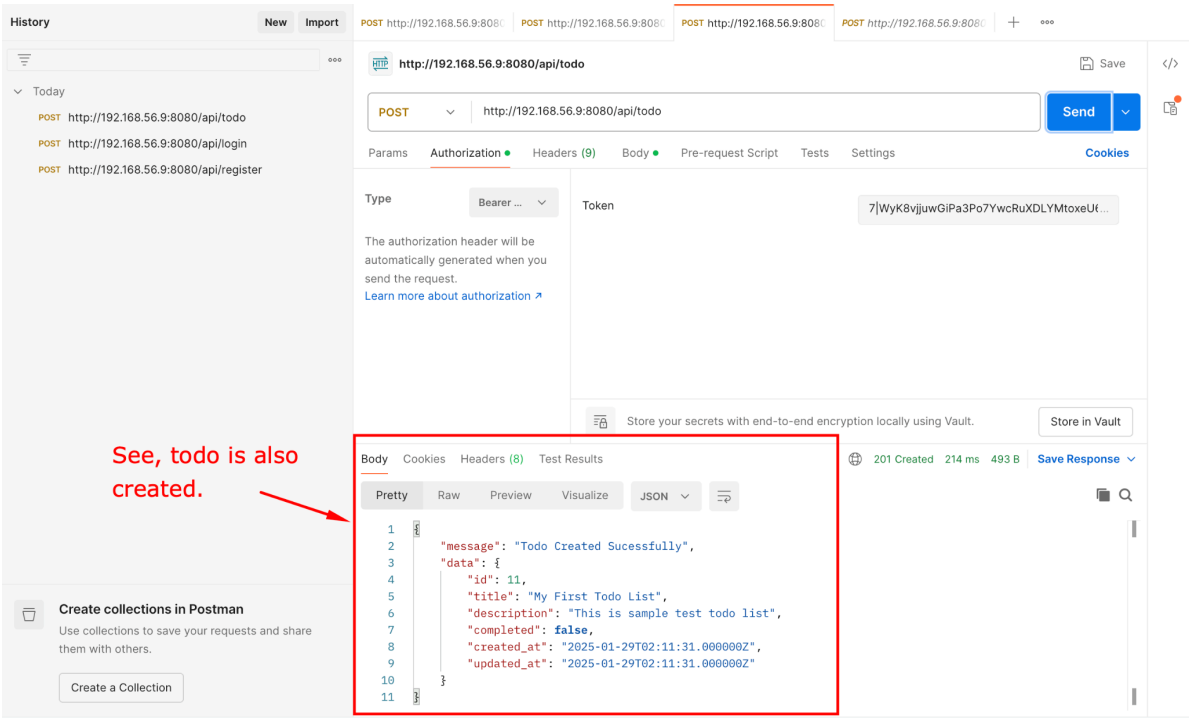
Inside postman:



And go to Authorization section in postman:



As result:

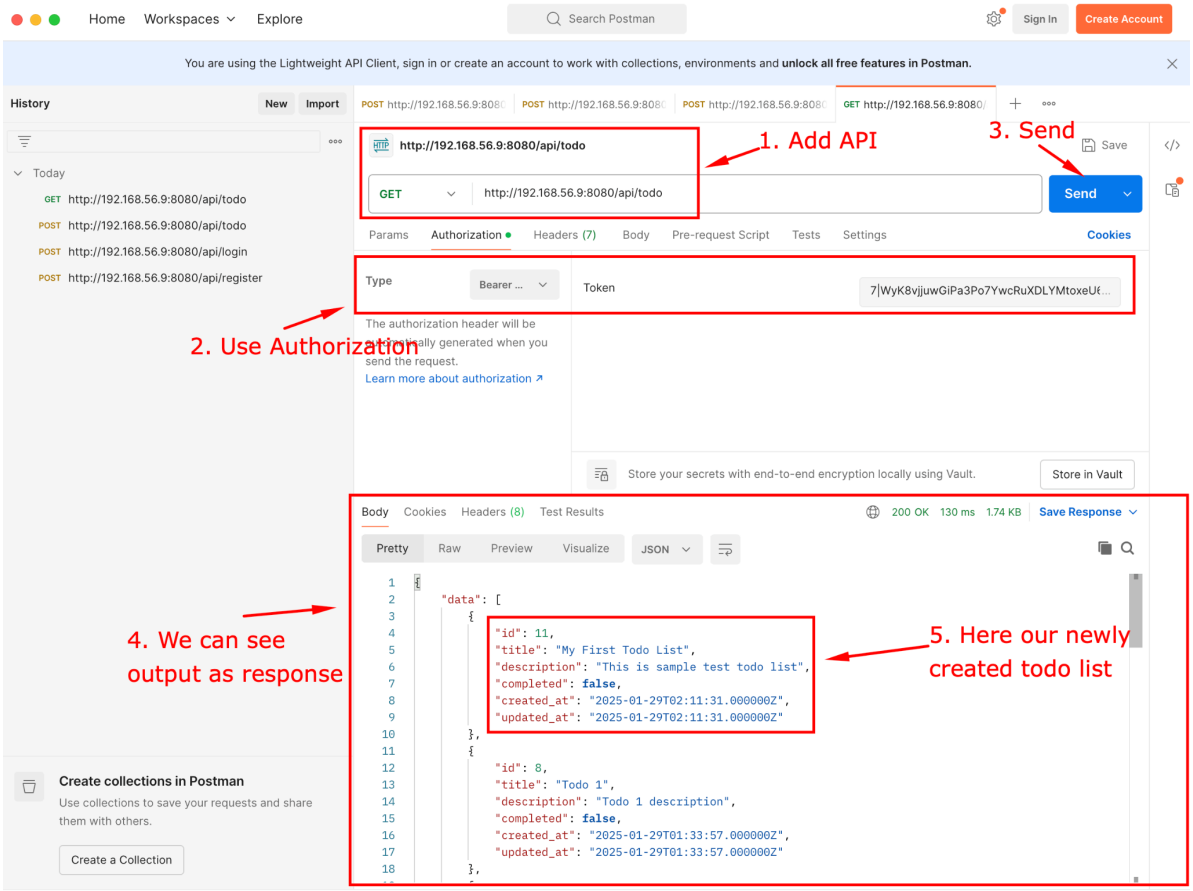


Successfully created todo.

4. List To-Do Items
Request:

```
GET http://192.168.56.9:8080/api/todo

Authorization: Bearer YOUR_ACCESS_TOKEN
Content-Type: application/json
```

Inside Postman:

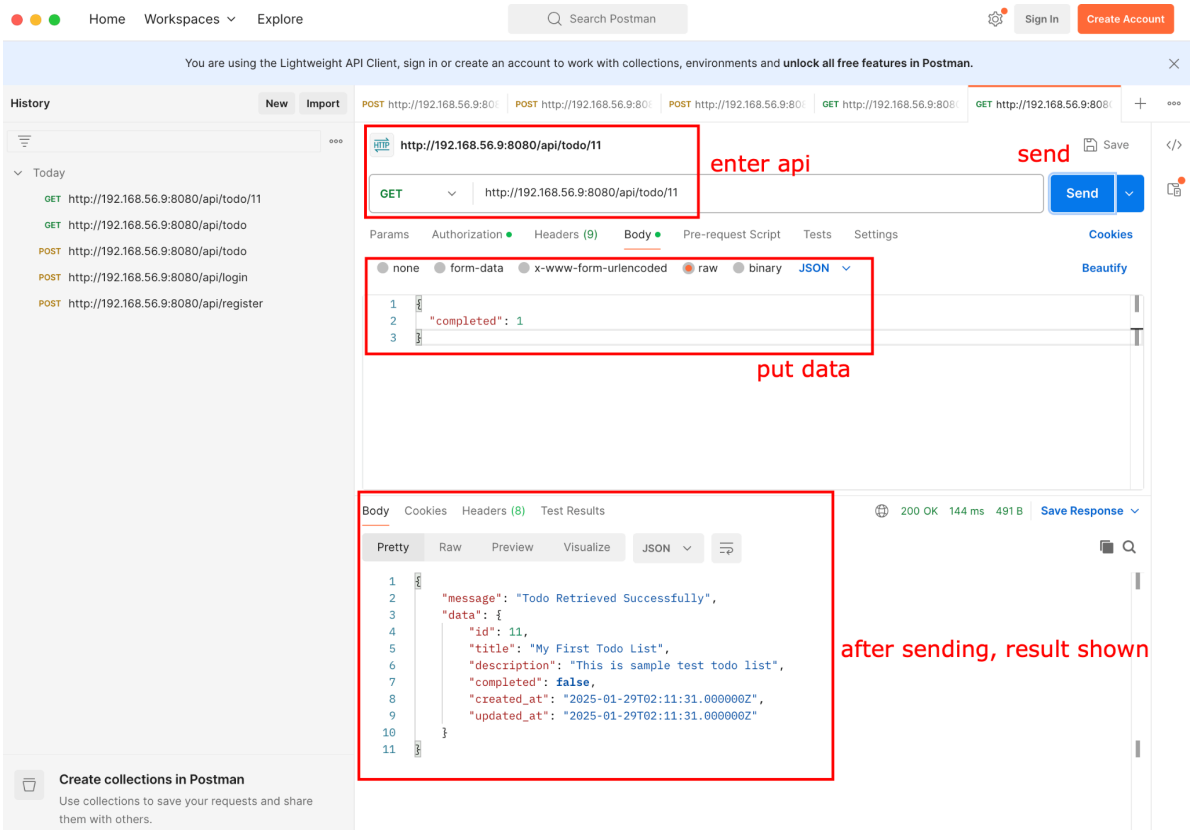


This how we can get our data from DB to real time while login process.

5. To-Do Item Detail
Request:

```
GET http://192.168.56.9:8080/api/todo/11

Authorization: Bearer YOUR_ACCESS_TOKEN
Content-Type: application/json

{
  "completed": 1
}
```

Inside Postman:

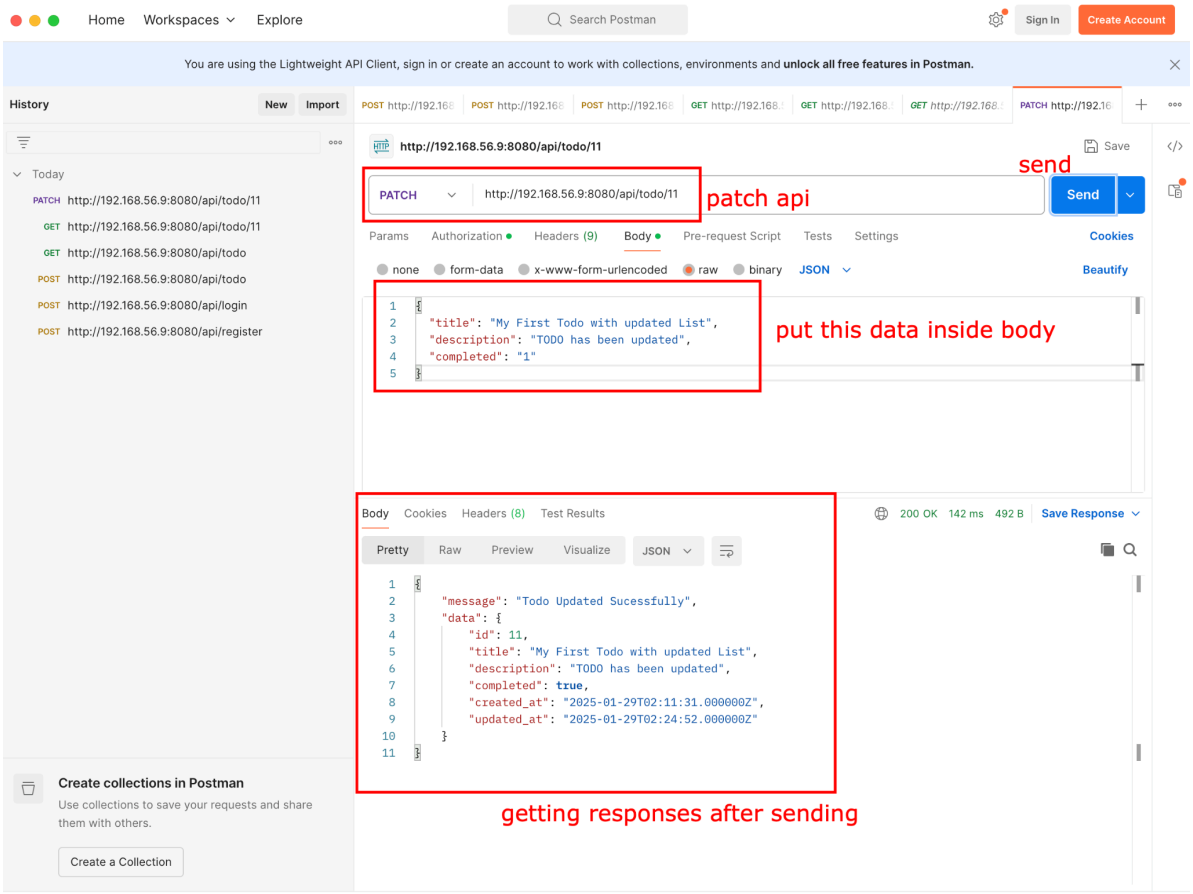


We can retrieve detail of created todo list, here 11 is todo id while it is creating we can see its id above.

6. Update To-Do Item
Request:

```
PATCH http://192.168.56.9:8080/api/todo/11
Authorization: Bearer YOUR_ACCESS_TOKEN
Content-Type: application/json

{
  "title": "My First Todo with updated List",
  "description": "TODO has been updated",
  "completed": "1"
}
```

Use Authorization with bearer token as previously used way.

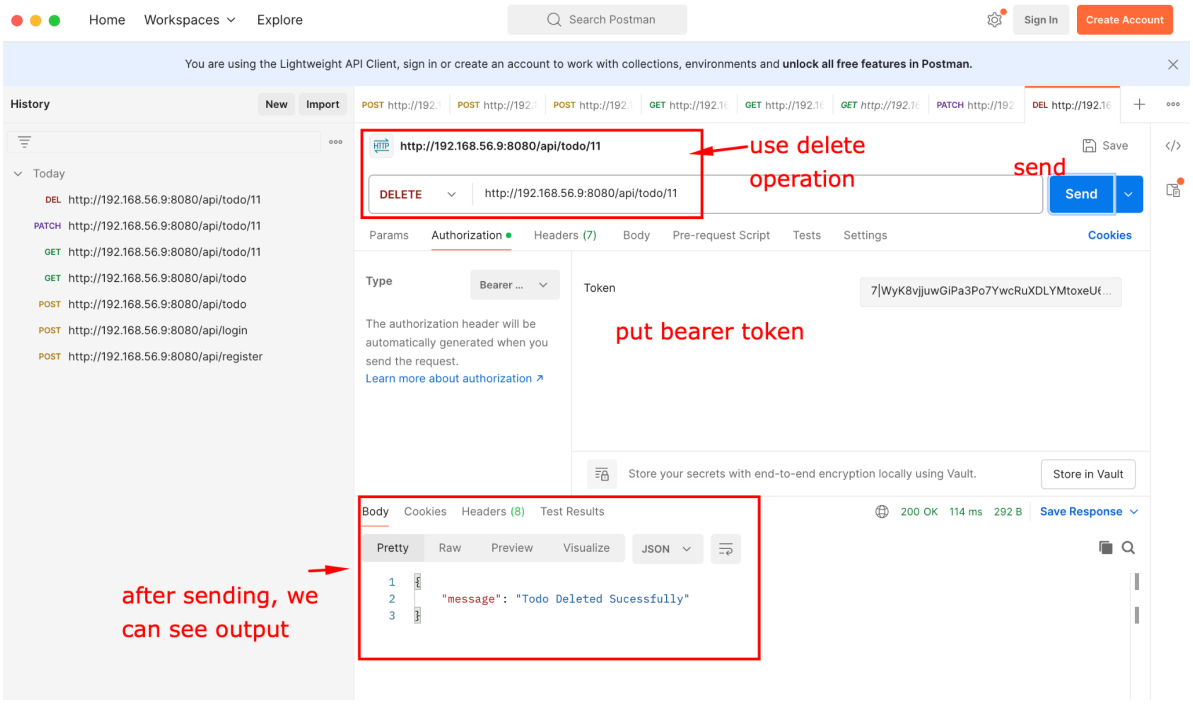


Now, we can see PATCH operation also performed successfully.

7. Delete To-Do Item
Request:

```
DELETE http://192.168.56.9:8080/api/todo/11
Authorization: Bearer YOUR_ACCESS_TOKEN
Content-Type: application/json
```

Inside Postman:



Delete Operation also performed successfully.

For deployment

1. **Remote Server Setup**: Use Vagrant, Multipass, or similar virtualization tools to set up a remote server.

   Let's Setup Remote server with vagrant:

   Vagrant is a tool for managing virtual machines. It allows you to create reproducible environments. I am going to use vagrant for server setup.

   I have created vagrant file along with ansible file for automated provision inside my server which is running:

   Vagrantfile: Is a type of file which manages and automate the entire vm. We can use it for provision, run, stop, & destroy the vms.

   Also Ansible file used for provision and setup all dependencies while provisioning server.

```
bishnuthapa@Bishnus-MacBook-Pro Ansible-server % vagrant up
Bringing machine 'UbuntuVM' up with 'virtualbox' provider...
==> UbuntuVM: Importing base box 'ubuntu/jammy64'...
==> UbuntuVM: Matching MAC address for NAT networking...
==> UbuntuVM: Checking if box 'ubuntu/jammy64' version '20241002.0.0' is up to date...
==> UbuntuVM: Setting the name of the VM: Ansible-server_UbuntuVM_1738158197592_40982
==> UbuntuVM: Clearing any previously set network interfaces...
==> UbuntuVM: Preparing network interfaces based on configuration...
    UbuntuVM: Adapter 1: nat
    UbuntuVM: Adapter 2: hostonly
==> UbuntuVM: Forwarding ports...
    UbuntuVM: 22 (guest) => 2222 (host) (adapter 1)
==> UbuntuVM: Running 'pre-boot' VM customizations...
==> UbuntuVM: Booting VM...
==> UbuntuVM: Waiting for machine to boot. This may take a few minutes...
    UbuntuVM: SSH address: 127.0.0.1:2222
    UbuntuVM: SSH username: vagrant
    UbuntuVM: SSH auth method: private key
    UbuntuVM: Warning: Connection reset. Retrying...
    UbuntuVM: Warning: Remote connection disconnect. Retrying...
    UbuntuVM:
    UbuntuVM: Vagrant insecure key detected. Vagrant will automatically replace
    UbuntuVM: this with a newly generated keypair for better security.
    UbuntuVM:
    UbuntuVM: Inserting generated public key within guest...
    UbuntuVM: Removing insecure key from the guest if it's present...
    UbuntuVM: Key inserted! Disconnecting and reconnecting using new SSH key...
==> UbuntuVM: Machine booted and ready!
==> UbuntuVM: Checking for guest additions in VM...
    UbuntuVM: The guest additions on this VM do not match the installed version of
    UbuntuVM: VirtualBox! In most cases this is fine, but in rare cases it can
    UbuntuVM: prevent things such as shared folders from working properly. If you see
    UbuntuVM: shared folder errors, please make sure the guest additions within the
    UbuntuVM: virtual machine match the version of VirtualBox you have installed on
    UbuntuVM: your host and reload your VM.
    UbuntuVM:
    UbuntuVM: Guest Additions Version: 6.0.0 r127566
    UbuntuVM: VirtualBox Version: 7.1
==> UbuntuVM: Configuring and enabling network interfaces...
==> UbuntuVM: Mounting shared folders...
    UbuntuVM: /Users/bishnuthapa/Desktop/Practice/Vms/Ansible-server => /vagrant
==> UbuntuVM: Running provisioner: ansible...
Vagrant gathered an unknown Ansible version:


and falls back on the compatibility mode '1.8'.

Alternatively, the compatibility mode can be specified in your Vagrantfile:
```

Here we can see, ansible is working:

```
    UbuntuVM: Running ansible-playbook...

PLAY [all] *********************************************************************

TASK [Gathering Facts] *********************************************************
[WARNING]: Platform linux on host UbuntuVM is using the discovered Python
interpreter at /usr/bin/python3.10, but future installation of another Python
interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-
core/2.18/reference_appendices/interpreter_discovery.html for more information.
ok: [UbuntuVM]

TASK [Install dependencies] ****************************************************
changed: [UbuntuVM]

TASK [Install Composer] ********************************************************
changed: [UbuntuVM]

TASK [Install Composer globally] ***********************************************
changed: [UbuntuVM]

TASK [Configure Nginx] *********************************************************
changed: [UbuntuVM]

RUNNING HANDLER [restart nginx] ************************************************
changed: [UbuntuVM]

PLAY RECAP *********************************************************************
UbuntuVM                   : ok=6    changed=5    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

bishnuthapa@Bishnus-MacBook-Pro Ansible-server %
```

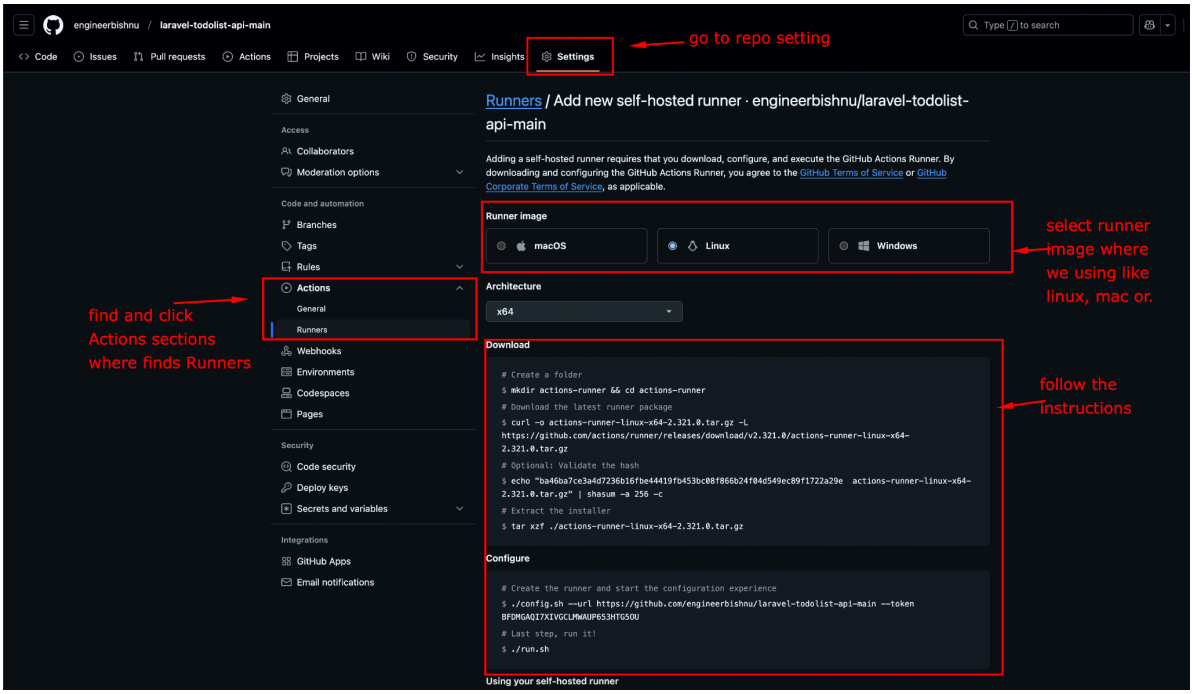
Now, Ansible setup the rest of pre-requisites for me which makes my server is ready to deploy laravel app with the help of self-hosted runner.

2. **Self-Hosted Runner**: Set up a GitHub or Bitbucket self-hosted runner.

Let's Setup self-hosted runner:

Steps:

a. Go to repository's setting tab and inside setting find Actions.
b. Follow the instructions shown in image below:



We can see here, self-hosted runner setup steps. Let’s do inside remote server.

```
bishnuthapa@Bishnus-MacBook-Pro Ansible-server % vagrant ssh
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 5.15.0-130-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Wed Jan 29 13:54:58 UTC 2025

  System load:             0.0
  Usage of /:              5.8% of 38.70GB
  Memory usage:            58%
  Swap usage:              0%
  Processes:               100
  Users logged in:         0
  IPv4 address for enp0s3: 10.0.2.15
  IPv6 address for enp0s3: fd00::7e:b0ff:fe35:270c


Expanded Security Maintenance for Applications is not enabled.

15 updates can be applied immediately.
8 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

2 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

New release '24.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


Last login: Wed Jan 29 13:46:13 2025 from 10.0.2.2
vagrant@ubuntu-jammy:~$
```

We enter the remote vm recently created. Let’s setup self-hosted runner inside it:

Follow the given instruction above:

Download

```
# Create a folder
$ mkdir actions-runner && cd actions-runner
# Download the latest runner package
$ curl -o actions-runner-linux-x64-2.321.0.tar.gz -L https://github.com/actions/runner/releases/download/v2.321.0/actions-runner-linux-x64-2.321.0.tar.gz
# Optional: Validate the hash
$ echo "ba46ba7ce3a4d7236b16fbe44419fb453bc08f866b24f04d549ec89f1722a29e  actions-runner-linux-x64-2.321.0.tar.gz" | shasum -a 256 -c
# Extract the installer
$ tar xzf ./actions-runner-linux-x64-2.321.0.tar.gz
```

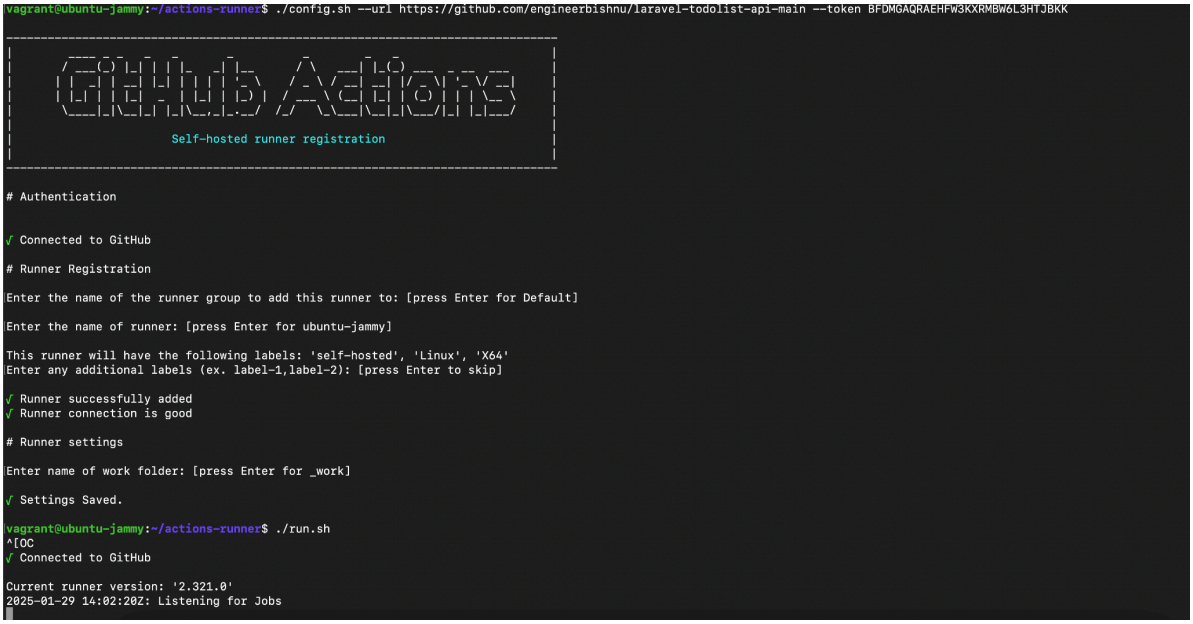
Configure

```
# Create the runner and start the configuration experience
$ ./config.sh --url https://github.com/engineerbishnu/laravel-todolist-api-main --token BFDMGAQI7XIVGCLMWAUP653HTG5OU
# Last step, run it!
$ ./run.sh
```

Using your self-hosted runner

```
# Use this YAML in your workflow file for each job
runs-on: self-hosted
```

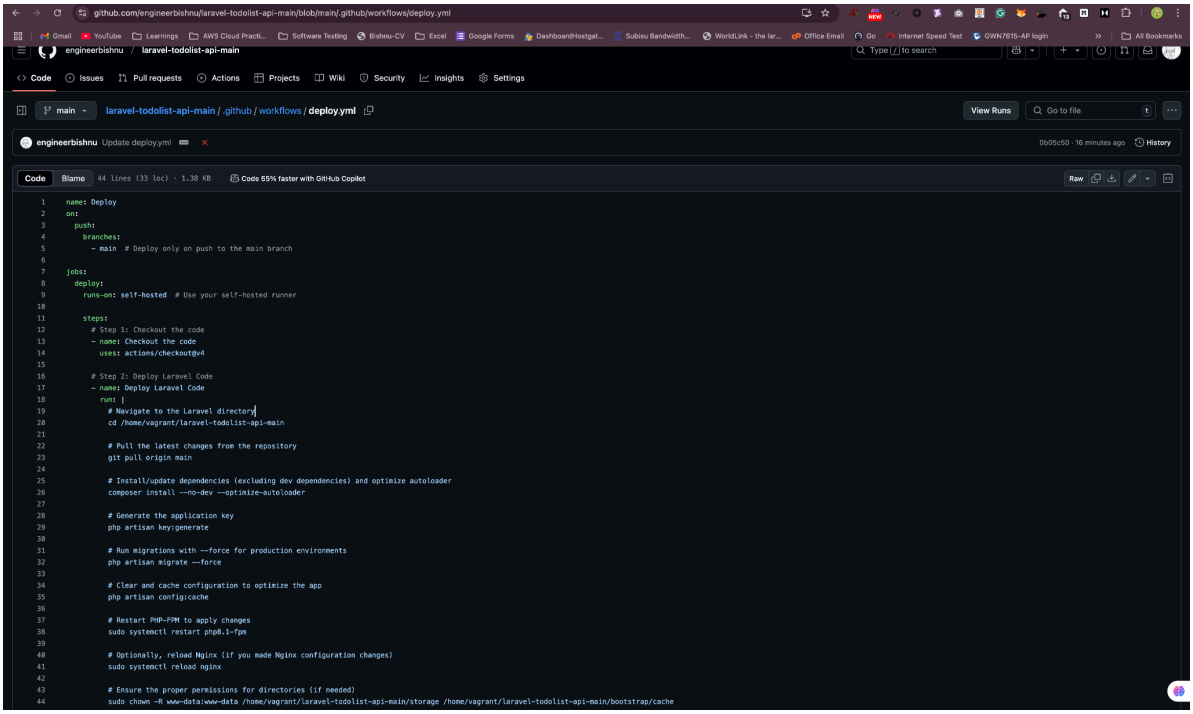
These are the steps to perform inside remote vm.



We can see, our github self-hosted runner is setup and running. Also listening for incoming deployments from github.

After setup self-hoster runner, let's create cicd pipeline, using github action for CI and self hosted runner for CD part.

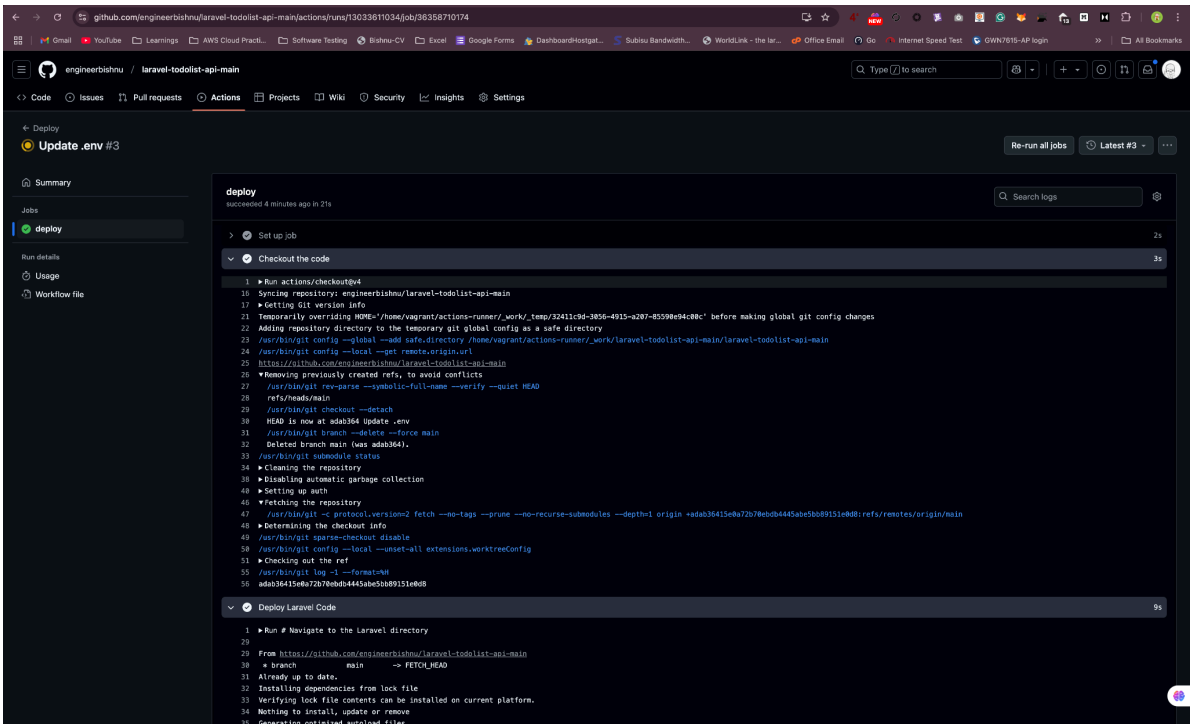
3. **CI/CD Pipeline**: Implement a CI/CD pipeline with GitHub Actions or Bitbucket Pipelines, including build, test, and deployment stages. Use the self-hosted runner for this pipeline.



```
name: Deploy
on:
  push:
    branches:
      - main  # Deploy only on push to the main branch

jobs:
  deploy:
    runs-on: self-hosted  # Use your self-hosted runner

    steps:
      # Step 1: Checkout the code
      - name: Checkout the code
        uses: actions/checkout@v4

      # Step 2: Deploy Laravel Code
      - name: Deploy Laravel Code
        run: |
          # Navigate to the Laravel directory
          cd /home/vagrant/laravel-todolist-api-main

          # Pull the latest changes from the repository
          git pull origin main

          # Install/update dependencies (excluding dev dependencies) and optimize autoloader
          composer install --no-dev --optimize-autoloader

          # Generate the application key
          php artisan key:generate

          # Run migrations with --force for production environments
          php artisan migrate --force

          # Clear and cache configuration to optimize the app
          php artisan config:cache

          # Restart PHP-FPM to apply changes
          sudo systemctl restart php8.1-fpm

          # Optionally, reload Nginx (if you made Nginx configuration changes)
          sudo systemctl reload nginx

          # Ensure the proper permissions for directories (if needed)
          sudo chown -R www-data:www-data /home/vagrant/laravel-todolist-api-main/storage /home/vagrant/laravel-todolist-api-main/bootstrap/cache
```

I have created .github/workflow/deploy,yml file which tracks all changes on every push command whether developer push the code, CI happened and it will do automatically deploy the deployment server.

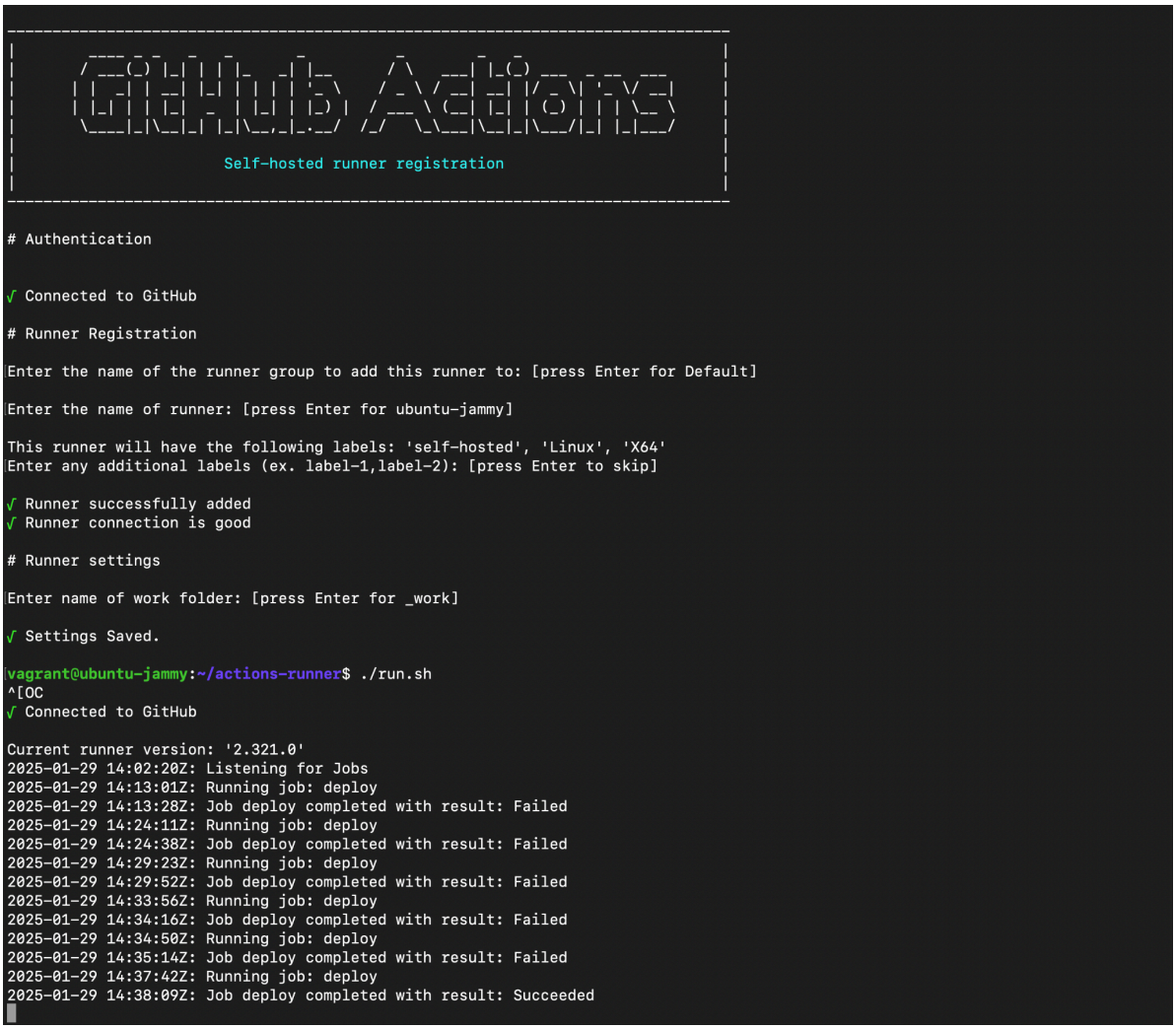
After push, it will trigger action:



```
Deploy
Update .env #3
Re-run all jobs
Latest #3
Summary
Jobs
deploy
Run details
Usage
Workflow file
deploy
succeeded 4 minutes ago in 21s
Search logs
Set up job 2s
Checkout the code 3s
  Run actions/checkout@v4
  Syncing repository: engineerbishnu/laravel-todolist-api-main
  Getting Git version info
  Temporarily overriding HOME='/home/vagrant/actions-runner/_work/_temp/32411c9d-3056-4915-a207-85590e94c00c' before making global git config changes
  Adding repository directory to the temporary git global config as a safe directory
  /usr/bin/git config --global --add safe.directory /home/vagrant/actions-runner/_work/laravel-todolist-api-main/laravel-todolist-api-main
  /usr/bin/git config --local --get remote.origin.url
  https://github.com/engineerbishnu/laravel-todolist-api-main
  Removing previously created refs, to avoid conflicts
    /usr/bin/git rev-parse --symbolic-full-name --verify --quiet HEAD
    refs/heads/main
    /usr/bin/git checkout --detach
    HEAD is now at adab364 Update .env
    /usr/bin/git branch --delete --force main
    Deleted branch main (was adab364).
  /usr/bin/git submodule status
  Cleaning the repository
  Disabling automatic garbage collection
  Setting up auth
  Fetching the repository
    /usr/bin/git -c protocol.version=2 fetch --no-tags --prune --no-recurse-submodules --depth=1 origin +adab36415e0a72b70ebdb4445abe5bb89151e0d8:refs/remotes/origin/main
  Determining the checkout info
  /usr/bin/git sparse-checkout disable
  /usr/bin/git config --local --unset-all extensions.worktreeConfig
  Checking out the ref
  /usr/bin/git log -1 --format=%H
  adab36415e0a72b70ebdb4445abe5bb89151e0d8
Deploy Laravel Code 9s
  Run # Navigate to the Laravel directory

  From https://github.com/engineerbishnu/laravel-todolist-api-main
   * branch            main       -> FETCH_HEAD
  Already up to date.
  Installing dependencies from lock file
  Verifying lock file contents can be installed on current platform.
  Nothing to install, update or remove
```

Above, I have done workflow file changed and push, it trigger the action.
Action will automatically do CI and CD accordingly to deployment or remote server.

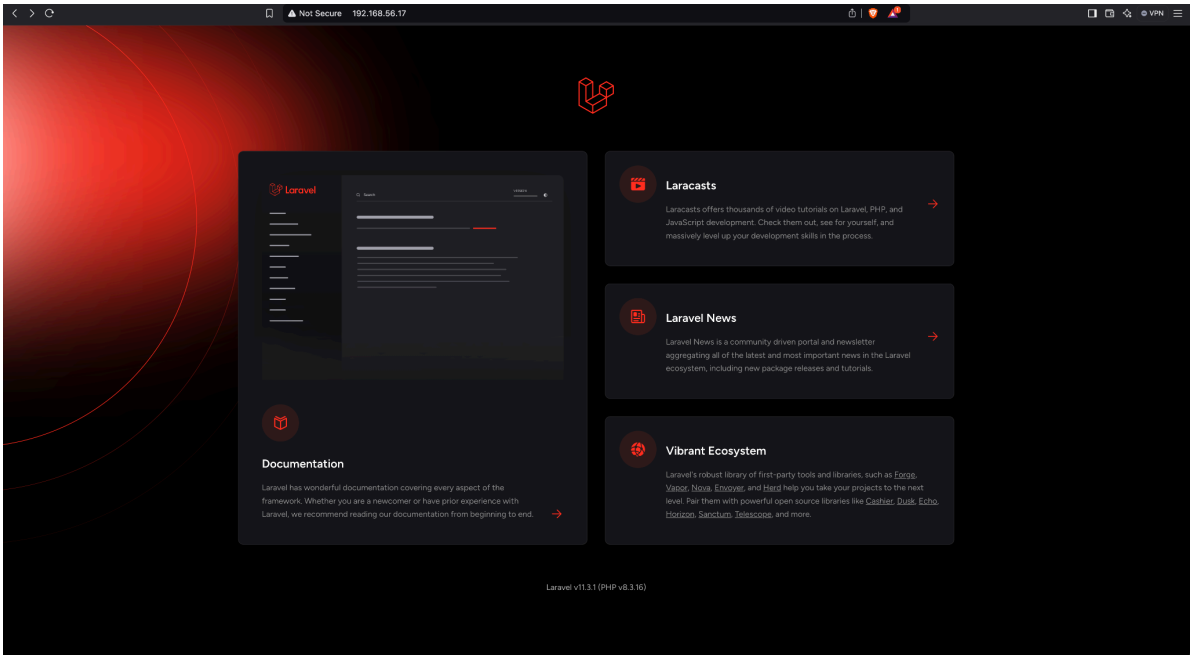
After 4 attempts, due to minor issues and fails, I have fixed, I am able to successfully deploy laravel application in my remote vm without using Docker.



4. **Deployment**: Deploy the application to a remote server without Docker, only using a web server (e.g., Nginx) and a PHP processor (e.g., PHP-FPM).
    1. This deployment should occur from your GitHub or Bitbucket pipeline using the self-hosted runner.

We can see above, laravel application is deployed successfully.

We can verify it by browsing vm ip accordingly.



5. **Infrastructure Automation**: Write Infrastructure as Code (IaC) or an Ansible playbook to automate the setup of the remote server, including all dependencies.

   → For automation, I have used ansible playbook provided in repository.

   For infra setup:

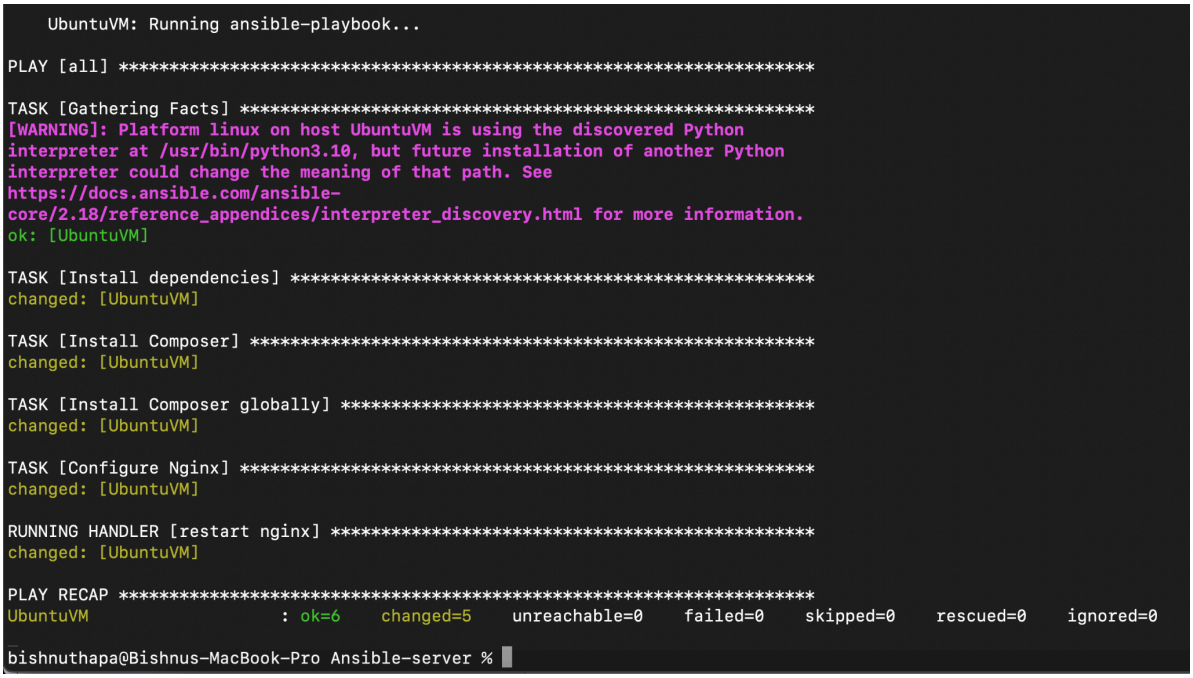
```
bishnuthapa@Bishnus-MacBook-Pro ansible_server % ls
Vagrantfile     ansible
bishnuthapa@Bishnus-MacBook-Pro ansible_server % cat Vagrantfile
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/jammy64"
  config.vm.box_version = "20241002.0.0"

  config.vm.provider "virtualbox" do |vb|
    vb.memory = "1024"
    vb.cpus = "1"
  end

  config.vm.define "UbuntuVM" do |subconfig|
    subconfig.vm.network :private_network, type: "dhcp"
  end

  config.vm.provision "ansible" do |ansible|
    ansible.playbook = "ansible/setup.yml"
  end
end
```

And for automation with ansible:

```
bishnuthapa@Bishnus-MacBook-Pro ansible_server % cat ansible/setup.yml
- hosts: all
  become: yes
  tasks:
    # Add necessary package dependencies
    - name: Add PHP 8.3 PPA
      apt_repository:
        repo: ppa:ondrej/php
        state: present
        update_cache: yes

    # Install necessary dependencies, including PHP 8.3 specific packages
    - name: Install dependencies
      apt:
        name:
          - nginx
          - php8.3-fpm
          - php8.3-mysql
          - php8.3-redis
          - php8.3-xml
          - php8.3-mbstring
          - php8.3-bcmath
          - php8.3-gd
          - mysql-server
          - redis-server
          - git
          - unzip
        update_cache: yes

    # Install Composer
    - name: Download Composer installer
      get_url:
        url: https://getcomposer.org/installer
        dest: /tmp/composer-installer.php
        mode: '0755'

    # Install Composer globally
    - name: Install Composer globally
      command: php /tmp/composer-installer.php --install-dir=/usr/local/bin --filename=composer
      args:
        creates: /usr/local/bin/composer

    # Cleanup: Remove composer installer after installation
    - name: Remove Composer installer
      file:
        path: /tmp/composer-installer.php
        state: absent

  handlers:
    # Restart nginx service if needed
    - name: restart nginx
      service:
        name: nginx
        state: restarted

bishnuthapa@Bishnus-MacBook-Pro ansible_server %
```

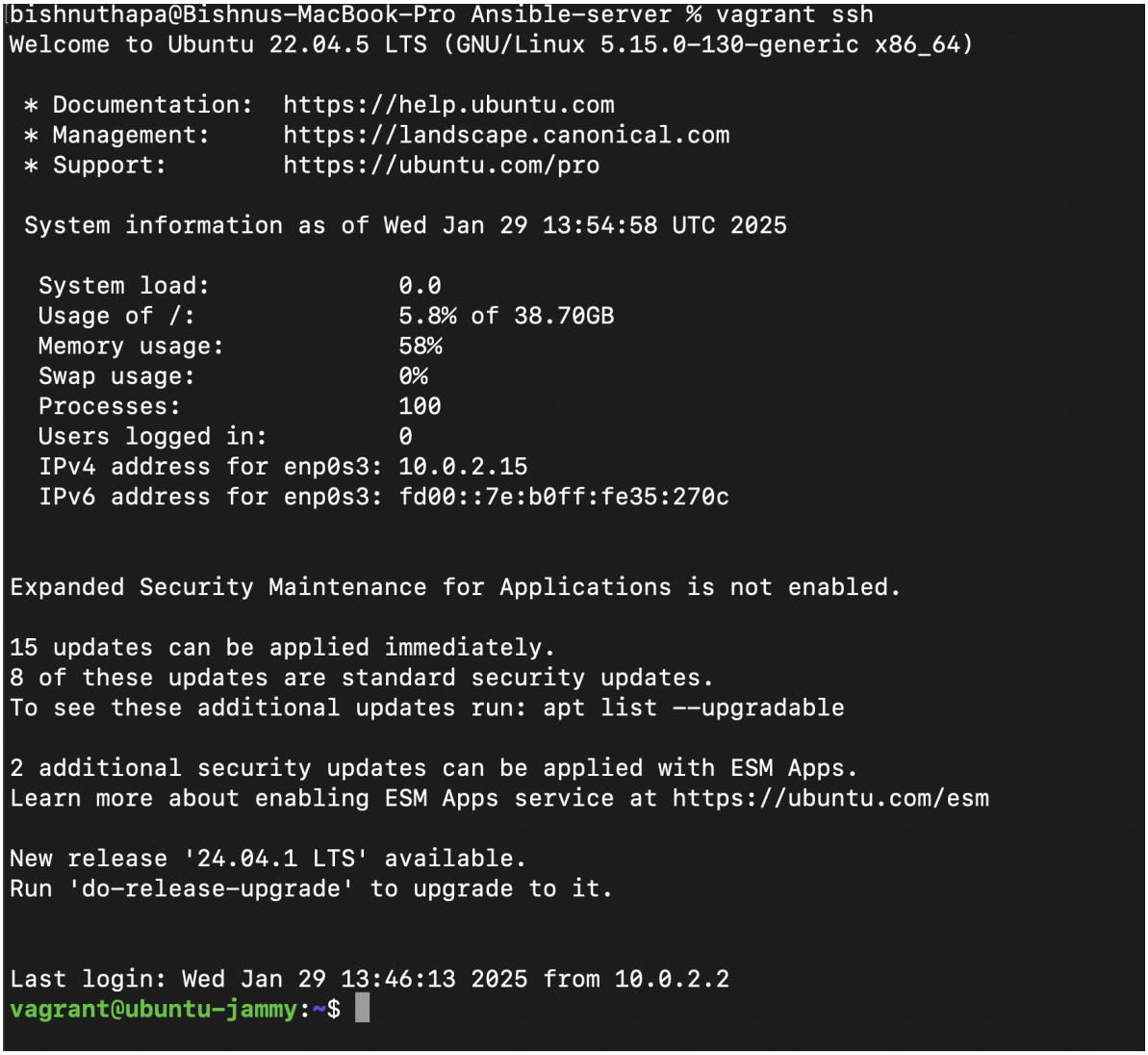
This is how automate setup and prepare server env local or remote server.

After provision we can see:



Where vagrant up brings up fresh server and ansible do configure pre-requisites.

Now we can do vagrant ssh to enter the remote server:



6. **Documentation**: Update the README or add a new deployment README with detailed local setup instructions, and include deployment documentation covering CI/CD and remote server setup.

   Updated README.md file, and I have two docs files for API testing and lab setup.