

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"]
```

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"

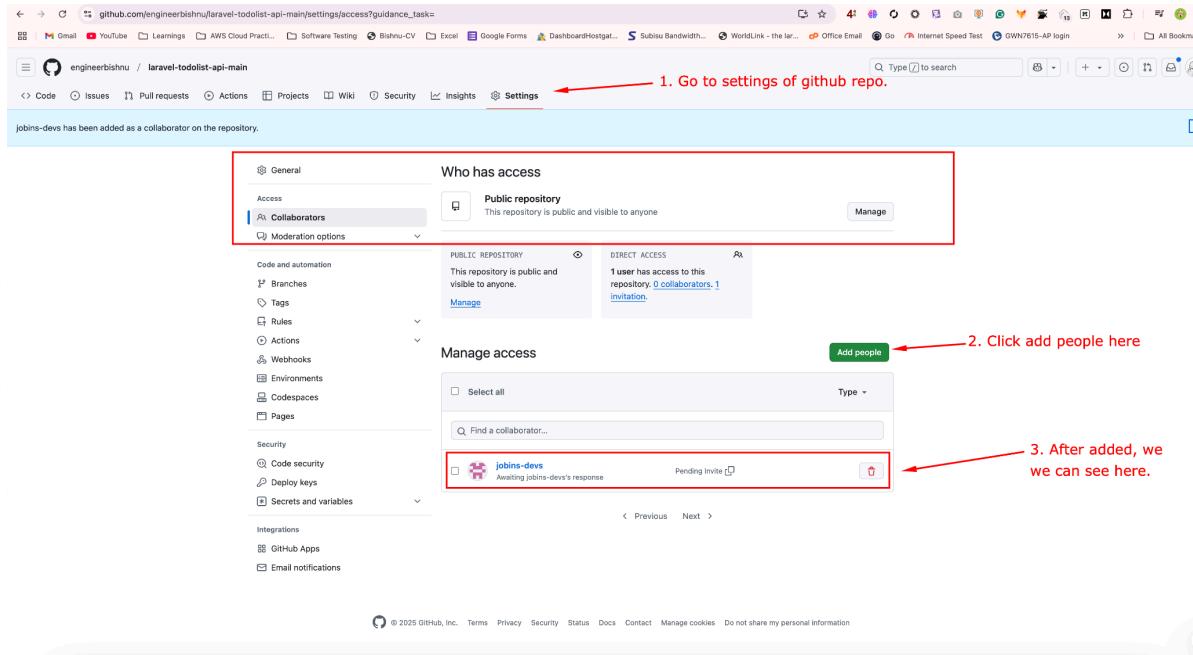
volumes:
  db_data:

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.

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.

1. enter api

2. choose as json

3. put data in body

4. send

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

we can see the response

```

{
  "data": {
    "id": 5,
    "name": "bishnu",
    "email": "bishnu@tester.com"
  },
  "token": "6|dsM9XXYK02mr5aHdiKwsZucFcAkZL9xNzxyxEiHm38e4545d"
}

```

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:

The screenshot shows the Postman interface with a red box highlighting the request details. Inside the red box, there are three numbered steps: 1. Use login api, 2. Use data inside body, and 3. Send. Below the request details, the response body is shown in a red box with step 4. Get output as shown. The response body is a JSON object:

```
1
2   "data": {
3     "id": 5,
4     "name": "bishnu",
5     "email": "bishnu@tester.com"
6   },
7   "token": "7|WYK8vjjuw6iPa3Po7YwcRuXDLYMTtoxeU6aJy3jEc42b825f9"
```

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:

The screenshot shows the Postman interface with a request configuration for a POST API endpoint. The URL is set to `http://192.168.56.9:8080/api/todo`. The 'Body' tab is selected, showing a JSON payload with the following content:

```
1
2   "title": "My First Todo List",
3   "description": "This is sample test todo list",
4   "completed": "0"
5 }
```

Annotations with red arrows point to specific parts of the interface:

1. Put todo API: Points to the URL field.
2. Put data inside body: Points to the JSON payload in the Body tab.

And go to Authorization section in postman:

The screenshot shows the Postman interface with the Authorization section open. The 'Type' dropdown is set to 'Bearer ...'. A token value, `7|WyK8vjuwGiPa3Po7YwcRuXDLYMtoxeUi...`, is pasted into the 'Token' input field. The 'Send' button is highlighted with a red arrow.

Annotations with red arrows point to specific steps:

1. Choose auth: Points to the 'Type' dropdown.
2. Type Bearer Token: Points to the 'Token' input field.
3. Paste token here: Points to the token value in the input field.
4. Send: Points to the 'Send' button.

As result:

See, todo is also created.

```

1
2   "message": "Todo Created Sucessfully",
3   "data": {
4     "id": 11,
5     "title": "My First Todo List",
6     "description": "This is sample test todo list",
7     "completed": false,
8     "created_at": "2025-01-29T02:11:31.000000Z",
9     "updated_at": "2025-01-29T02:11:31.000000Z"
10
11

```

Successfully created todo.

4. List To-Do Items

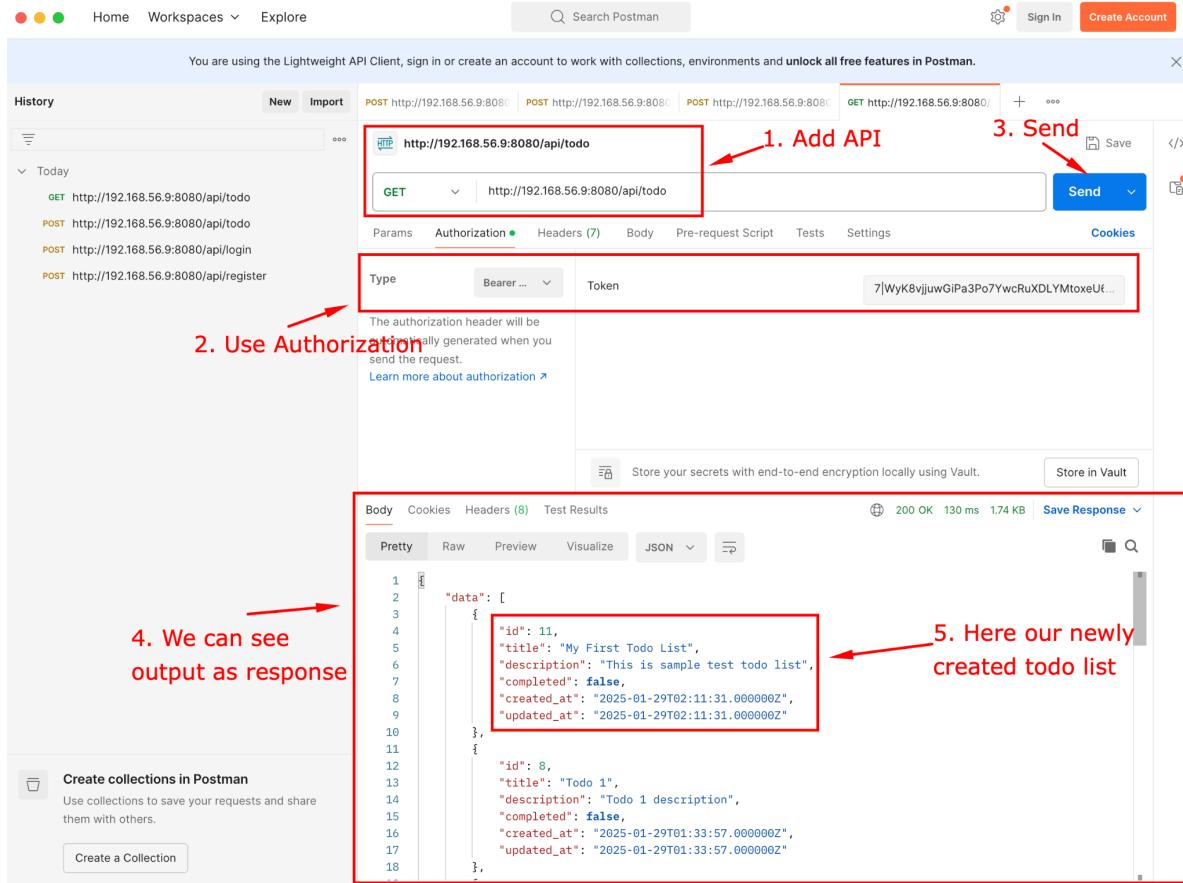
Request:

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

```
Authorization: Bearer YOUR_ACCESS_TOKEN
```

```
Content-Type: application/json
```

Inside Postman:



You are using the Lightweight API Client, sign in or create an account to work with collections, environments and unlock all free features in Postman.

History

New Import

POST http://192.168.56.9:8080/api/todo **POST** http://192.168.56.9:8080/api/login **POST** http://192.168.56.9:8080/api/register **GET** http://192.168.56.9:8080/api/todo

1. Add API

2. Use Authorization

3. Send

4. We can see output as response

5. Here our newly created todo list

The authorization header will be automatically generated when you send the request.
Learn more about authorization >

Store your secrets with end-to-end encryption locally using Vault. **Save Response**

Body Cookies Headers (8) Test Results **200 OK 130 ms 1.74 KB**

Pretty Raw Preview Visualize JSON

```

1   "data": [
2     {
3       "id": 11,
4       "title": "My First Todo List",
5       "description": "This is sample test todo list",
6       "completed": false,
7       "created_at": "2025-01-29T02:11:31.000000Z",
8       "updated_at": "2025-01-29T02:11:31.000000Z"
9     },
10    {
11      "id": 8,
12      "title": "Todo 1",
13      "description": "Todo 1 description",
14      "completed": false,
15      "created_at": "2025-01-29T01:33:57.000000Z",
16      "updated_at": "2025-01-29T01:33:57.000000Z"
17    }
18  ]

```

Create collections in Postman
Use collections to save your requests and share them with others.

Create a Collection

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:

The screenshot shows the Postman interface with a PUT request to `http://192.168.56.9:8080/api/todo/11`. The request body contains the JSON object `{"completed": 1}`. The response pane shows a successful 200 OK status with the message "Todo Retrieved Successfully" and the retrieved todo item.

```

{
  "message": "Todo Retrieved Successfully",
  "data": {
    "id": 11,
    "title": "My First Todo List",
    "description": "This is sample test todo list",
    "completed": false,
    "created_at": "2025-01-29T02:11:31.000000Z",
    "updated_at": "2025-01-29T02:11:31.000000Z"
  }
}

```

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.

The screenshot shows the Postman interface with a successful PATCH operation. The URL is `http://192.168.56.9:8080/api/todo/11`. The request body contains JSON data:

```

1
2   "title": "My First Todo with updated List",
3   "description": "TODO has been updated",
4   "completed": "1"
5

```

The response body shows the updated todo item:

```

1
2   "message": "Todo Updated Sucessfully",
3   "data": {
4     "id": 11,
5     "title": "My First Todo with updated List",
6     "description": "TODO has been updated",
7     "completed": true,
8     "created_at": "2025-01-29T02:11:31.000000Z",
9     "updated_at": "2025-01-29T02:24:52.000000Z"
10   }
11

```

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:

The screenshot shows the Postman interface with a red box highlighting the DELETE request for `http://192.168.56.9:8080/api/todo/11`. A red arrow points to the 'DELETE' button with the text 'use delete operation'. Another red arrow points to the 'Body' tab of the response section with the text 'after sending, we can see output'. The response body contains the JSON object: `1 "message": "Todo Deleted Sucessfully"`.

Delete Operation also performed successfully.

For deployment

- 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 %
```

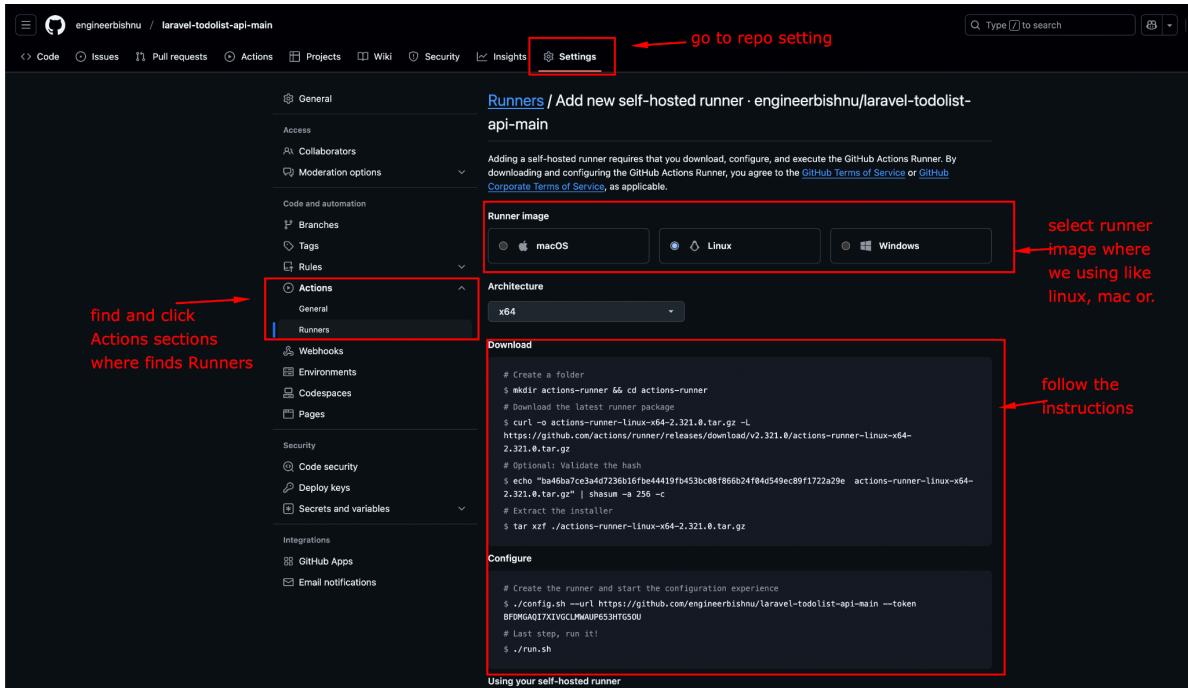
Now, Ansible setup the rest of pre-requisites for me which makes my server is ready to deploy laravel app with the help of self-hosted runner.

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  
BFDMGQAQI7XIVGCLMWAUP653HTG50U  
  
# 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.

```
Last login: Wed Jan 29 13:46:13 2025 from 10.0.2.2  
vagrant@ubuntu-jammy:~$ mkdir actions-runner && cd actions-runner  
vagrant@ubuntu-jammy:~/actions-runner$ 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.t  
ar.gz  
% Total    % Received % Xferd  Average Speed   Time     Time Current  
          Dload Upload   Total Spent    Left Speed  
 0      0      0      0      0      0      0      0      0      0  
100  119M  119M  0      0  120M  0:01:36 0:01:36 0  1486K  
vagrant@ubuntu-jammy:~/actions-runner$ echo "ba46ba7ce3a4d7236b16fbe44419fb453bc08f866b24f04d549ec89f1722a29e" actions-runner-linux-x64-2.321.0.tar.gz" | shasum -a 256 -c  
actions-runner-linux-x64-2.321.0.tar.gz: OK  
vagrant@ubuntu-jammy:~/actions-runner$ tar xzf ./actions-runner-linux-x64-2.321.0.tar.gz  
vagrant@ubuntu-jammy:~/actions-runner$ ls  
actions-runner-linux-x64-2.321.0.tar.gz  bin  config.sh  env.sh  externals  run-helper.cmd.template  run-helper.sh.template  run.sh  safe_sleep.sh
```

```
vagrant@ubuntu-jammy:~/actions-runner$ ./config.sh --url https://github.com/engineerbishnu/laravel-todolist-api-main --token BFDMGQRAEHFW3KXRMBW6L3HTJBKK
  _/\_/_/_/  _/\_/_/_/  _/\_/_/_/  _/\_/_/_/
 / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \
\   \   \   \   \   \   \   \   \   \   \   \   \   \
 \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_
Self-hosted runner registration

# 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 ubuntu-jammy]
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 connection is good

# Runner settings

Enter name of work folder: [press Enter for _work]
✓ Settings Saved.

[vagrant@ubuntu-jammy:~/actions-runner$ ./run.sh
^[[C
✓ Connected to GitHub

Current runner version: '2.321.0'
2025-01-29 14:02:26: Listening for Jobs
```

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:
    - name: Checkout the code
      uses: actions/checkout@v4
    - 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 chmod -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
succeeded 4 minutes ago in 21s
  > Set up job
  > Checkout the code
    ▶ Run actions/checkout@v4
    1 From https://github.com/actions/checkout@v4
    2 Using Node.js v14.17.0
    3 Getting Git version info
    4 Temporarily overriding HOME='/home/vagrant/actions-runner/_work/_temp/32411c9d-3056-4915-a207-8559e94c00c' before making global git config changes
    5 Adding repository directory to the temporary git global config as a safe directory
    6 /usr/bin/git config --global --add safe.directory /home/vagrant/actions-runner/_work/laravel-todolist-api-main/laravel-todolist-api-main
    7 /usr/bin/git config --global core.autocrlf input
    8 /usr/bin/git config --global core.eol lf
    9 /usr/bin/git config --global core.ignorecase true
    10 /usr/bin/git config --global core.loglevel 0
    11 /usr/bin/git config --global core.symlinks false
    12 /usr/bin/git config --global http.sslVerify false
    13 /usr/bin/git config --global http.sslCert /etc/ssl/certs/ca-certificates.crt
    14 /usr/bin/git config --global http.sslKey /etc/ssl/private/ca-certificates.key
    15 /usr/bin/git config --global http.sslCAInfo /etc/ssl/certs/ca-certificates.crt
    16 Removing previously created refs, to avoid conflicts
    17 /usr/bin/git rev-parse --symbolic-full-name --verify --quiet HEAD
    18 refs/heads/main
    19 /usr/bin/git checkout --detach
    20 HEAD is now at adab36415e8a72b78ebdb4445abe5bb89151e8d8 refs/remotes/origin/main
    21 Already up to date.
    22 /usr/bin/git sparse-checkout disable
    23 /usr/bin/git config --local --unset-all extensions.worktreeConfig
    24 /usr/bin/git config --local --unset-all extensions.gitignore
    25 /usr/bin/git log -1 --format=%H
    26 adab36415e8a72b78ebdb4445abe5bb89151e8d8
  > Deploy Laravel Code
    ▶ Run # Navigate to the Laravel directory
    27 From https://github.com/engineerbishnu/laravel-todolist-api-main
    28 * branch    main      -> FETCH_HEAD
    29 Already up to date.
    30 No new file or packages from lock file
    31 Verifying lock file contents can be installed on current platform.
    32 Nothing to install, update or remove
    33 Generation completed in 1.000ms

```

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.

```
Self-hosted runner registration

# 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 ubuntu-jammy]
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 connection is good

# Runner settings
Enter name of work folder: [press Enter for _work]
✓ Settings Saved.

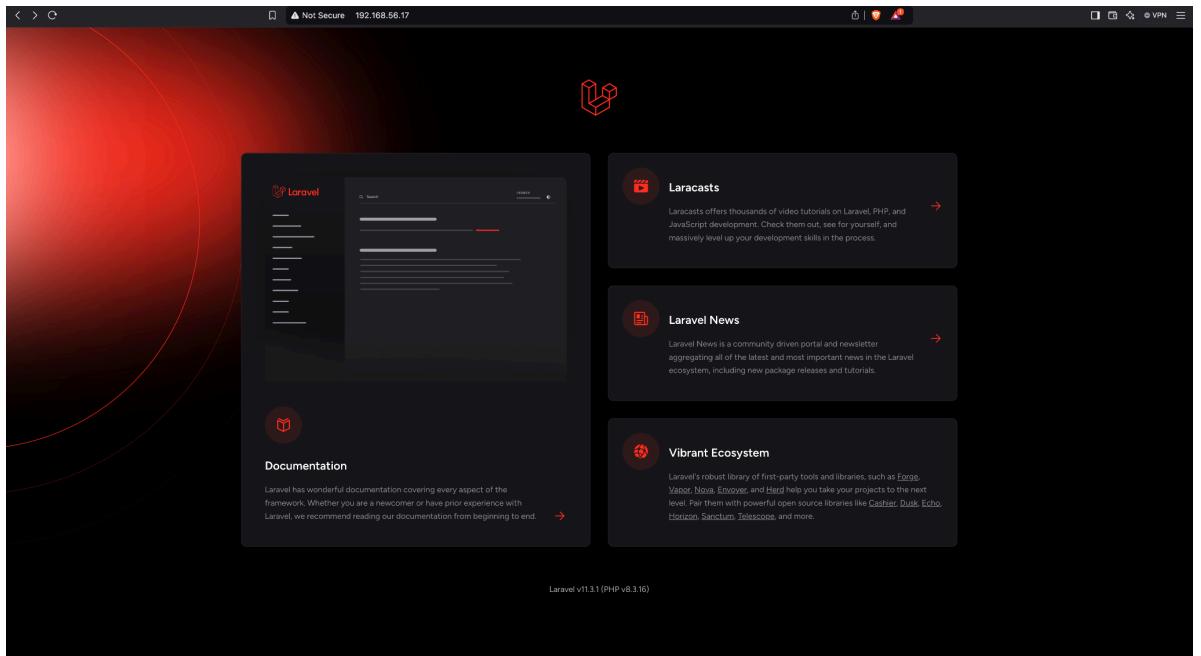
[vagrant@ubuntu-jammy:~/actions-runner$ ./run.sh
^[OC
✓ Connected to GitHub

Current runner version: '2.321.0'
2025-01-29 14:02:0Z: Listening for Jobs
2025-01-29 14:13:0Z: Running job: deploy
2025-01-29 14:13:28Z: Job deploy completed with result: Failed
2025-01-29 14:24:11Z: Running job: deploy
2025-01-29 14:24:38Z: Job deploy completed with result: Failed
2025-01-29 14:29:23Z: Running job: deploy
2025-01-29 14:29:52Z: Job deploy completed with result: Failed
2025-01-29 14:33:56Z: Running job: deploy
2025-01-29 14:34:16Z: Job deploy completed with result: Failed
2025-01-29 14:34:50Z: Running job: deploy
2025-01-29 14:35:14Z: Job deploy completed with result: Failed
2025-01-29 14:37:42Z: Running job: deploy
2025-01-29 14:38:09Z: Job deploy completed with result: Succeeded
```

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:

```
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 %
```

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

Now we can do vagrant ssh to enter the 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:~$ █
```

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.