Contents

LIVT Stack + MySQL + User Authentication + SSL + Docker 2

- Prerequisites (Windows) 2
- Step 1: Project Initialization (LIVT Stack) 2
- Step 2: Database & Backend Logic 2
- Step 3: Frontend Views (Vue + Tailwind) 7
- Step 4: Self-Signed SSL (Windows) 11
- Step 5: Docker Configuration 12
- Step 6: Build & Upload to Docker Hub (Windows) 16
- Step 8: Test Container in Windows 17
- Step 9: Deploy to Ubuntu 20
  - Install Docker Engine in Ubuntu 20
  - Getting the container 21
- Step 10: CI / CD with GitHub Actions 22

LIVT Stack + MySQL + User Authentication + SSL + Docker

Prerequisites (Windows)

- **Docker Desktop** installed and running.
- **PHP** and **Composer** installed.
- **Node.js** installed.

Step 1: Project Initialization (LIVT Stack)

We will use **Laravel Breeze** to scaffold the LIVT stack (Laravel, Inertia, Vue, Tailwind) automatically.

1. Open your Terminal (PowerShell/Command Prompt).
2. Create the project:

```
composer create-project laravel/laravel livt-app
cd livt-app
```

3. Install Breeze and choose the stack:

```
composer require laravel/breeze --dev
php artisan breeze:install
```

   - **Which stack?** Select Vue with Inertia.
   - **Dark mode?** Optional (Choose No for simplicity).
   - **Testing framework?** PHPUnit.

4. If you are having dependency errors because of vite/vue/tailwind version mismatches, use the following command instead (which will manually install the latest stable and compatible versions of the three).

```
npm install -D vite@^5.4.0 laravel-vite-plugin@^1.0.0 @vitejs/plugin-vue@^5.0.0
@tailwindcss/vite@^4.0.0
```

Step 2: Database & Backend Logic

We will create a simple CRUD system for "Notes".

Setup .env file

```
...
DB_CONNECTION=mysql
DB_HOST=db          <-- CHANGE THIS (Matches service name in docker-compose.yml)
DB_PORT=3306
DB_DATABASE=livt_db
DB_USERNAME=livt_user
DB_PASSWORD=secret_password
...
```

A. Database Migration
Run

```
php artisan make:model Note -m
```

database/migrations/xxxx_xx_xx_create_notes_table.php

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('notes', function (Blueprint $table) {
            $table->id();
            $table->foreignId('user_id')->constrained()->onDelete('cascade'); // Link note to user
            $table->string('title');
            $table->text('content');
            $table->timestamps();
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('notes');
    }
};
```

B. The Model

app/Models/Note.php

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Note extends Model
{
    use HasFactory;

    protected $fillable = ['title', 'content', 'user_id'];
    // Relationship to User
    public function user()
    {
        return $this->belongsTo(User::class);
    }
}
```

<u>app/Models/User.php</u>

```
<?php

namespace App\Models;

// use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Illuminate\Database\Eloquent\Relations\HasMany;

class User extends Authenticatable
{
    /** @use HasFactory<\Database\Factories\UserFactory> */
    use HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var list<string>
     */
    protected $fillable = [
        'name',
        'email',
        'password',
    ];

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var list<string>
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];
```

```
  /**
   * Get the attributes that should be cast.
   *
   * @return array<string, string>
   */
  protected function casts(): array
  {
    return [
      'email_verified_at' => 'datetime',
      'password' => 'hashed',
    ];
  }

  public function notes(): HasMany
  {
    return $this->hasMany(Note::class);
  }
}
```

C. The Controller

Run

```
php artisan make:controller NoteController --resource
```

app/Http/Controllers/NoteController.php

```
<?php

namespace App\Http\Controllers;

use App\Models\Note;
use Illuminate\Http\Request;
use Inertia\Inertia;
use Illuminate\Support\Facades\Auth;

class NoteController extends Controller
{
  // READ (List)
  public function index()
  {
    // Get notes only for the logged-in user
    $notes = Note::where('user_id', Auth::id())->latest()->get();
    return Inertia::render('Notes/Index', ['notes' => $notes]);
  }



  // CREATE (Form)
```

```
public function create()
{
   return Inertia::render('Notes/Create');
}

// STORE (Save to DB)
public function store(Request $request)
{
   $validated = $request->validate([
      'title' => 'required|string|max:255',
      'content' => 'required|string',
   ]);

   $request->user()->notes()->create($validated);

   return redirect()->route('notes.index');
}

// EDIT (Form)
public function edit(Note $note)
{
   if ($note->user_id !== Auth::id()) { abort(403); }
   return Inertia::render('Notes/Edit', ['note' => $note]);
}

// UPDATE (Save changes)
public function update(Request $request, Note $note)
{
   if ($note->user_id !== Auth::id()) { abort(403); }

   $validated = $request->validate([
      'title' => 'required|string|max:255',
      'content' => 'required|string',
   ]);

   $note->update($validated);

   return redirect()->route('notes.index');
}

// DELETE
public function destroy(Note $note)
{
   if ($note->user_id !== Auth::id()) { abort(403); }
   $note->delete();
```

```
        return redirect()->route('notes.index');
    }
}
```

D. Routes (Protected by Middleware)

routes/web.php

```
<?php

use App\Http\Controllers\ProfileController;
use App\Http\Controllers\NoteController;
use Illuminate\Foundation\Application;
use Illuminate\Support\Facades\Route;
use Inertia\Inertia;

Route::get('/', function () {
    return Inertia::render('Welcome', [
        'canLogin' => Route::has('login'),
        'canRegister' => Route::has('register'),
    ]);
});

Route::get('/dashboard', function () {
    return Inertia::render('Dashboard');
})->middleware(['auth', 'verified'])->name('dashboard');

// Protected Routes Group
Route::middleware('auth')->group(function () {
    Route::get('/profile', [ProfileController::class, 'edit'])->name('profile.edit');
    Route::patch('/profile', [ProfileController::class, 'update'])->name('profile.update');
    Route::delete('/profile', [ProfileController::class, 'destroy'])->name('profile.destroy');

    // CRUD Resource Route
    Route::resource('notes', NoteController::class);
});

require __DIR__.'/auth.php';
```

Step 3: Frontend Views (Vue + Tailwind)

Create a folder
resources/js/Pages/Notes

A. Index (List View)

resources/js/Pages/Notes/Index.vue

```
<script setup>
import AuthenticatedLayout from '@/Layouts/AuthenticatedLayout.vue';
import { Head, Link, useForm } from '@inertiajs/vue3';

defineProps({ notes: Array });

const form = useForm({});

const deleteNote = (id) => {
  if (confirm("Are you sure you want to delete this note?")) {
    form.delete(route('notes.destroy', id));
  }
};
</script>

<template>
  <Head title="My Notes" />

  <AuthenticatedLayout>
    <template #header>
      <h2 class="font-semibold text-xl text-gray-800 leading-tight">My Notes</h2>
    </template>

    <div class="py-12">
      <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
        <div class="mb-6">
          <Link :href="route('notes.create')" class="bg-indigo-600 hover:bg-indigo-700 text-white px-4 py-
2 rounded shadow">
            + Create New Note
          </Link>
        </div>

        <div class="bg-white overflow-hidden shadow-sm sm:rounded-lg">
          <div class="p-6 text-gray-900">
            <table class="min-w-full divide-y divide-gray-200">
              <thead>
                <tr>
                  <th class="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase tracking-
wider">Title</th>
```

```
                        <th class="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase tracking-
wider">Content</th>
                        <th class="px-6 py-3 text-right text-xs font-medium text-gray-500 uppercase tracking-
wider">Actions</th>
                      </tr>
                    </thead>
                    <tbody class="bg-white divide-y divide-gray-200">
                      <tr v-for="note in notes" :key="note.id">
                        <td class="px-6 py-4 whitespace-nowrap font-bold">{{ note.title }}</td>
                        <td class="px-6 py-4">{{ note.content }}</td>
                        <td class="px-6 py-4 whitespace-nowrap text-right text-sm font-medium">
                          <Link :href="route('notes.edit', note.id)" class="text-indigo-600 hover:text-indigo-900
mr-4">Edit</Link>
                          <button @click="deleteNote(note.id)" class="text-red-600 hover:text-red-
900">Delete</button>
                        </td>
                      </tr>
                      <tr v-if="notes.length === 0">
                        <td colspan="3" class="px-6 py-4 text-center text-gray-500">No notes found.</td>
                      </tr>
                    </tbody>
                  </table>
                </div>
              </div>
            </div>
          </div>
  </AuthenticatedLayout>
</template>
```

B. Create View

resources/js/Pages/Notes/Create.vue

```
<script setup>
import AuthenticatedLayout from '@/Layouts/AuthenticatedLayout.vue';
import { Head, useForm, Link } from '@inertiajs/vue3';

const form = useForm({
  title: '',
  content: ''
});

const submit = () => {
  form.post(route('notes.store'));
};
</script>
```

```
<template>
  <Head title="Create Note" />
  <AuthenticatedLayout>
    <template #header>
      <h2 class="font-semibold text-xl text-gray-800 leading-tight">Create Note</h2>
    </template>
    <div class="py-12">
      <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
        <div class="bg-white overflow-hidden shadow-sm sm:rounded-lg p-6">
          <form @submit.prevent="submit">
            <div class="mb-4">
              <label class="block font-medium text-sm text-gray-700">Title</label>
              <input v-model="form.title" type="text" class="border-gray-300 focus:border-indigo-500
focus:ring-indigo-500 rounded-md shadow-sm w-full" />
              <div v-if="form.errors.title" class="text-red-500 text-sm mt-1">{{ form.errors.title }}</div>
            </div>
            <div class="mb-4">
              <label class="block font-medium text-sm text-gray-700">Content</label>
              <textarea v-model="form.content" class="border-gray-300 focus:border-indigo-500
focus:ring-indigo-500 rounded-md shadow-sm w-full h-32"></textarea>
              <div v-if="form.errors.content" class="text-red-500 text-sm mt-1">{{ form.errors.content
}}</div>
            </div>
            <div class="flex items-center gap-4">
              <button type="submit" class="bg-indigo-600 text-white px-4 py-2 rounded">Save
Note</button>
              <Link :href="route('notes.index')" class="text-gray-600 hover:text-gray-900">Cancel</Link>
            </div>
          </form>
        </div>
      </div>
    </div>
  </AuthenticatedLayout>
</template>
```

C. Edit View

resources/js/Pages/Notes/Edit.vue

```
<script setup>
import AuthenticatedLayout from '@/Layouts/AuthenticatedLayout.vue';
import { Head, useForm, Link } from '@inertiajs/vue3';

const props = defineProps({ note: Object });
const form = useForm({
  title: props.note.title,
  content: props.note.content
});

const submit = () => {
  form.put(route('notes.update', props.note.id));
};
</script>

<template>
  <Head title="Edit Note" />
  <AuthenticatedLayout>
    <template #header>
      <h2 class="font-semibold text-xl text-gray-800 leading-tight">Edit Note</h2>
    </template>
    <div class="py-12">
      <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
        <div class="bg-white overflow-hidden shadow-sm sm:rounded-lg p-6">
          <form @submit.prevent="submit">
            <div class="mb-4">
              <label class="block font-medium text-sm text-gray-700">Title</label>
              <input v-model="form.title" type="text" class="border-gray-300 focus:border-indigo-500
focus:ring-indigo-500 rounded-md shadow-sm w-full" />
            </div>
            <div class="mb-4">
              <label class="block font-medium text-sm text-gray-700">Content</label>
              <textarea v-model="form.content" class="border-gray-300 focus:border-indigo-500
focus:ring-indigo-500 rounded-md shadow-sm w-full h-32"></textarea>
            </div>
            <div class="flex items-center gap-4">
              <button type="submit" class="bg-indigo-600 text-white px-4 py-2 rounded">Update
Note</button>
              <Link :href="route('notes.index')" class="text-gray-600 hover:text-gray-900">Cancel</Link>
            </div>
          </form>
        </div>
      </div>
    </div>
  </AuthenticatedLayout>
</template>
```

Step 4: Self-Signed SSL (Windows)

We need to generate certificates to allow Nginx to serve HTTPS.

1. Create a directory path in your project: mkdir -p docker/nginx/ssl
2. If you have Git Bash or OpenSSL installed on Windows, run this command in that folder. If not, install OpenSSL first.

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
-keyout docker/nginx/ssl/server.key \
-out docker/nginx/ssl/server.crt
```

(Just press Enter through the prompts).

3. You should now have server.key and server.crt inside livt-app/docker/nginx/ssl.

Step 5: Docker Configuration

A. Nginx Config

docker/nginx/conf.d/default.conf

```
server {
  listen 80;
  server_name localhost;
  return 301 https://$host$request_uri;
}


server {
  listen 443 ssl;
  server_name localhost;

  ssl_certificate /etc/nginx/ssl/server.crt;
  ssl_certificate_key /etc/nginx/ssl/server.key;

  root /var/www/public;
  index index.php index.html;

  location / {
    try_files $uri $uri/ /index.php?$query_string;
  }

  location ~ \.php$ {
    try_files $uri =404;
    fastcgi_split_path_info ^(.+\.php)(/.+)$;
    fastcgi_pass app:9000;
    fastcgi_index index.php;
    include fastcgi_params;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    fastcgi_param PATH_INFO $fastcgi_path_info;
  }
}
```

B. Nginx Dockerfile

Step 1: Create a Custom Nginx Dockerfile
Instead of using the raw nginx:alpine image, we will build a custom one that includes your SSL keys, configuration, and the compiled frontend assets.

Create a new file:
docker/nginx/Dockerfile

```
# STAGE 1: Build the Static Assets (Vue/Tailwind)
# We use Node to compile the CSS/JS so we don't depend on Windows files
FROM node:18-alpine as frontend_build
WORKDIR /app
COPY package*.json vite.config.js ./
RUN npm install
COPY resources ./resources
COPY public ./public
# This generates the files in /app/public/build
RUN npm run build

# STAGE 2: The Actual Nginx Server
FROM nginx:alpine

# Remove default configuration
RUN rm /etc/nginx/conf.d/default.conf

# Copy our custom config
COPY docker/nginx/conf.d/default.conf /etc/nginx/conf.d/default.conf

# Copy SSL Keys (Baked into the image)
COPY docker/nginx/ssl /etc/nginx/ssl

# Copy the Built Assets from Stage 1
COPY --from=frontend_build /app/public /var/www/public

# Copy the static files (index.php) from your source
# Note: Nginx doesn't run PHP, but it needs index.php to exist to pass it to the App
COPY public/index.php /var/www/public/index.php
COPY public/robots.txt /var/www/public/robots.txt
```

B. Dockerfile (For the App)
This is a multi-stage build. It installs PHP dependencies, then Node dependencies, builds the frontend assets, and cleans up.

Dockerfile (in project root)

```
# 1. Base Image
FROM php:8.2-fpm

# 2. Install System Dependencies & Node.js
RUN apt-get update && apt-get install -y \
   git curl libpng-dev libonig-dev libxml2-dev zip unzip \
   && curl -fsSL https://deb.nodesource.com/setup_20.x | bash - \
   && apt-get install -y nodejs

# 3. Install PHP Extensions
RUN docker-php-ext-install pdo_mysql mbstring exif pcntl bcmath gd

# 4. Get Composer
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer

# 5. Set Working Directory
WORKDIR /var/www

# 6. Copy Project Files
COPY . .

# 7. Install PHP Dependencies (Production)
RUN composer install --no-dev --optimize-autoloader

# 8. Build Frontend Assets (Vite/Tailwind)
RUN npm install
RUN npm run build

# 9. Set Permissions
# Allow www-data to write only to critical directories
RUN chown -R www-data:www-data /var/www/storage /var/www/bootstrap/cache

# Remove node_modules after building assets (Reduces image size significantly)
RUN rm -rf /var/www/node_modules
```

C. Docker Compose

docker-compose.yml

```
services:
 app:
  build: .
  image: johnreygoh/livt-app:latest # REPLACE THIS
  container_name: livt_app
  restart: unless-stopped
  working_dir: /var/www
  environment:
   APP_NAME: "LIVT App"
   APP_ENV: local
   APP_KEY: "base64:ytW/yfckQyYMDi8M+4hQgwp/vhs2LSs+SwBTZUBP4o0=" # Paste your key from .env
here for simplicity in this demo
   APP_DEBUG: "true"
   APP_URL: "https://localhost"
   DB_CONNECTION: mysql
   DB_HOST: db
   DB_PORT: 3306
   DB_DATABASE: livt_db
   DB_USERNAME: livt_user
   DB_PASSWORD: secret_password
  networks:
   - livt-net
  volumes:
   - static_assets:/var/www/public/build # App puts files here

 db:
  image: mysql:8.0
  container_name: livt_db
  restart: unless-stopped
  environment:
   MYSQL_DATABASE: livt_db
   MYSQL_USER: livt_user
   MYSQL_PASSWORD: secret_password
   MYSQL_ROOT_PASSWORD: root_password
  volumes:
   - db_data:/var/lib/mysql
  networks:
   - livt-net

 nginx:
  build:
   context: .
   dockerfile: docker/nginx/Dockerfile
  image: johnreygoh/livt-nginx:latest # <--- NEW IMAGE NAME
  container_name: livt_nginx
```

```
    restart: unless-stopped
    ports:
      - "80:80"
      - "443:443"
    # NOTICE: No volumes mapping local files!
    # Everything is inside the image now.
    depends_on:
      - app
    networks:
      - livt-net
    volumes:
      - static_assets:/var/www/public/build # App puts files here



networks:
  livt-net:
    driver: bridge

volumes:
  db_data:
  static_assets:
```

D. Create a ".dockerignore" file

.dockerignore

```
/node_modules
/public/build
/vendor
.git
.env
```

Step 6: Build & Upload to Docker Hub (Windows)

1. **Login to Docker Hub:**

```
docker login
```

2. **Build the Image:**

Replace your_dockerhub_username with your actual ID.

```
docker build -t your_dockerhub_username/livt-app:latest .
```

3. **Push the Image:**

```
docker push your_dockerhub_username/livt-app:latest
```

Step 8: Test Container in Windows

1. Start Docker Desktop
2. Go to project root
3. Docker commands:

To build image (but not start it yet)

```
docker build -t your_dockerhub_username/livt-app:latest .
```

To start the container from the image

```
docker compose up
```

To build and start the container in a single command

```
docker compose up --build
```

To stop the container

```
docker compose down
```

To stop the container and delete volumes (if any)

```
docker compose down --v
```

To push image to docker hub

```
docker login
docker compose build
docker compose push
```

To pull image from docker hub and run in windows

1. Create a "Production" Folder Create a completely new folder somewhere else on your computer (e.g., C:\Users\You\Desktop\livt-production).

2. Create the docker-compose.yml for Deployment Inside this new folder, create a docker-compose.yml file. **Crucial Change:** We must **remove** the build: sections. We want Docker to download the images, not try to build them from missing source code.

```
services:
 app:
  # build: .  <-- REMOVE THIS
  image: johnreygoh/livt-app:latest
  container_name: livt_app
  restart: unless-stopped
  working_dir: /var/www
  environment:
   # You can hardcode these for prod, or keep using an env_file
```

```
    APP_NAME: "LIVT App"
    APP_ENV: production
    APP_KEY: "base64:..." # Use your real key here
    APP_DEBUG: "false"
    APP_URL: "https://localhost"
    DB_CONNECTION: mysql
    DB_HOST: db
    DB_PORT: 3306
    DB_DATABASE: livt_db
    DB_USERNAME: livt_user
    DB_PASSWORD: secret_password
   networks:
    - livt-net

  db:
   image: mysql:8.0
   container_name: livt_db
   restart: unless-stopped
   environment:
    MYSQL_DATABASE: livt_db
    MYSQL_USER: livt_user
    MYSQL_PASSWORD: secret_password
    MYSQL_ROOT_PASSWORD: root_password
   volumes:
    - db_data:/var/lib/mysql
   networks:
    - livt-net

  nginx:
   # build: ... <-- REMOVE THIS
   image: johnreygoh/livt-nginx:latest
   container_name: livt_nginx
   restart: unless-stopped
   ports:
    - "80:80"
    - "443:443"
   depends_on:
    - app
   networks:
    - livt-net

networks:
 livt-net:
  driver: bridge
```

```
volumes:
  db_data:
```

3. Create the .env file (Optional but Recommended) Although you *can* put environment variables directly in the compose file (as shown above), it is cleaner to copy your .env file to this new folder and change the app service to use env_file: .env.

- *Note: Since the images are baked, you do not need the source code, package.json, or composer.json.*

4. Pull the Images Open a terminal in this new folder and run:

```
docker compose pull
```

Docker will download johnreygoh/livt-app:latest and johnreygoh/livt-nginx:latest from Docker Hub.

5. Run the Application

```
docker compose up -d
```

6. Run Migrations (One time setup) Since this is a fresh database (new volume), you need to create the tables again.

```
docker compose exec app php artisan migrate
```

7. Test Go to https://localhost. You should see your application running perfectly, served entirely from the downloaded images, with no local source code required.

8. Common Error(s)

a. White screen
   (Browser) Page Inspection → console
   Look for errors like, Failed to load resource: server returned 404. Take note of the files that are missing.

   Check what you have:

   ```
   docker compose exec nginx ls -la /var/www/public
   docker compose exec app ls -la /var/www/
   ```

   The files might be created, but the Hash is different.

   - *Browser wants:* app-AB12.js
   - *Terminal shows:* app-CD34.js
   - The "App" image and "Nginx" image were built at different times or with slightly different source codes.
   - You must rebuild **both** images at the exact same time to ensure they match.

     ```
     docker compose build --no-cache
     docker compose up -d --force-recreate
     ```

b. Table not found

   Database tables haven't been created yet, run the migration command

```
docker compose exec app php artisan migrate
```

c. Could not connect to database

   If you executed the **docker compose up** command, it takes time before it gets ready. Wait for the line that says the MySQL *Server is ready to accept connections*.

Step 9: Deploy to Ubuntu

Now, SSH into your Ubuntu server.

Install Docker Engine in Ubuntu

To install Docker on Ubuntu, follow these steps. This process sets up the **Docker Engine** and adds your user to the Docker group so you don't have to type sudo for every command.

Step 1: Uninstall Old Versions

```
sudo apt-get remove docker docker-engine docker.io containerd runc
```

Step 2: Set Up the Repository
Update your package index and install prerequisites:

```
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg
```

Add Docker's official GPG key:

```
sudo install -m 0755 -d /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg

sudo chmod a+r /etc/apt/keyrings/docker.gpg
```

Add the repository to Apt sources:

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Step 3: Install Docker Engine
Update the package index again:

```
sudo apt-get update
```

Install Docker packages:

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

Step 4: Manage Docker as a Non-Root User (Recommended)
By default, you must use sudo to run Docker commands. To fix this:
Create the docker group (it may already exist):

```
sudo groupadd docker
```

Add your user to the group:

```
sudo usermod -aG docker $USER
```

Activate the changes: Log out and log back in, or run this command to refresh your group membership immediately:

```
newgrp docker
```

Step 5: Verify Installation
Run the "hello-world" image to verify everything is working correctly without sudo:

```
docker run hello-world
```

If you see a message saying ***"Hello from Docker!"****, your installation is successful and ready for deployment.*

Getting the container

1. **Prepare Directory**

```
mkdir livt-project
cd livt-project
```

2. **Create Docker Compose File**

Copy the docker-compose.yml content from Step 5C into a new file on Ubuntu.

Important:
Change

```
build: .
```

to just comment it out, or ensure it pulls the image: your_dockerhub_username/livt-app:latest.

3. **Setup Nginx Config:**

```
mkdir -p docker/nginx/conf.d
mkdir -p docker/nginx/ssl
```

- Create the docker/nginx/conf.d/default.conf file (paste content from Step 5A).
- **Securely copy** (SCP) your server.crt and server.key from Windows to this docker/nginx/ssl folder on Ubuntu.

4. **Run the Container:**

```
docker compose up -d
```

5. **Run Migrations:** Since the database is new, you must run the migration inside the container.

```
docker compose exec app php artisan migrate
```

Navigate to https://<your-ubuntu-ip> in your browser. Accept the self-signed certificate warning. You should see the Laravel landing page. You can now register, log in, and create notes.

Step 10: CI / CD with GitHub Actions

Right now, you are manually running docker compose build and docker push from your Windows computer. We will replace that manual work with **GitHub Actions**, so every time you push code to GitHub, it automatically builds your images and updates Docker Hub.

Step 1: Configure Docker Secrets in GitHub
To allow GitHub to upload to your Docker Hub account, you must give it your password securely.

1. Go to your Repository on **GitHub**.
2. Click **Settings** (top right tab).
3. In the left sidebar, scroll down to **Secrets and variables** > **Actions**.
4. Click **New repository secret**.
5. Add these two secrets:
   - **Name:** DOCKER_USERNAME
     - **Value:** johnreygoh (Your Docker Hub ID)
   - **Name:** DOCKER_PASSWORD
     - **Value:** Your Docker Hub Password (or Access Token).

Step 2: Create the Workflow File
GitHub Actions looks for instructions in a specific folder.

1. In your project root (on Windows), create this folder structure: .github/workflows
2. Inside that folder, create a file named **docker-publish.yml**.
3. Paste this content:

```
name: Build and Push to Docker Hub

on:
  push:
    branches: [ "main" ]
  # Allows you to run this workflow manually from the Actions tab
  workflow_dispatch:

jobs:
  build-and-push:
    runs-on: ubuntu-latest

    steps:
      # 1. Check out the code from the repo
      - name: Checkout Code
        uses: actions/checkout@v4

      # 2. Login to Docker Hub using the secrets you set up
      - name: Login to Docker Hub
        uses: docker/login-action@v3
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}

      # 3. Set up Docker Buildx (Required for advanced caching/building)
      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3

      # 4. Build and Push the APP Image
      - name: Build and Push App
        uses: docker/build-push-action@v5
        with:
          context: .
          file: ./Dockerfile
          push: true
          tags: ${{ secrets.DOCKER_USERNAME }}/livt-app:latest
          # Simple caching to speed up future builds
          cache-from: type=gha
          cache-to: type=gha,mode=max

      # 5. Build and Push the NGINX Image
      # Note: This one builds the frontend assets inside Docker, so it might take a minute
      - name: Build and Push Nginx
        uses: docker/build-push-action@v5
        with:
          context: .
          file: ./docker/nginx/Dockerfile
          push: true
          tags: ${{ secrets.DOCKER_USERNAME }}/livt-nginx:latest
          cache-from: type=gha
          cache-to: type=gha,mode=max
```

Step 3: Trigger the Action

1. **Commit and Push** this new file to GitHub:

```
git add .github/workflows/docker-publish.yml
git commit -m "Setup GitHub Actions"
git push origin main
```

2. Go to your GitHub repository in your browser.
3. Click the **Actions** tab.
4. You will see a workflow named "Build and Push to Docker Hub" running (yellow circle).
5. Click on it to watch the live logs.

When it turns Green: Your images on Docker Hub have been updated automatically. You didn't touch Docker on Windows at all!

How to Deploy the Updates (CD)

Now that Docker Hub has the new images, your Ubuntu server needs to download them.

The Manual Way (Simple):

1. SSH into Ubuntu.
2. Run:

```
docker compose pull  # Downloads the new layers from Hub
docker compose up -d # Replaces the containers
```

Here is how to automate the final step so your Ubuntu server updates itself immediately after a successful build.

To do this, we use an action called **appleboy/ssh-action** which allows GitHub to "SSH into" your server and run commands just like you would manually.

Step 1: Set up SSH Secrets on GitHub
GitHub needs permission to log into your Ubuntu server.

1. **Generate a new SSH Key (On Windows/Local):** You likely already have one, but it's safer to generate a dedicated one for GitHub.

```
ssh-keygen -t rsa -b 4096 -f github_deploy_key
```

(Press Enter for no passphrase).

2. **Add the Public Key to Ubuntu:** Copy the content of github_deploy_key.pub and append it to your server's authorized keys:

# On your Ubuntu server

```
nano ~/.ssh/authorized_keys
```

# Paste the content, save and exit.

3. **Add the Private Key to GitHub Secrets:**
   - Open github_deploy_key (the one *without* the extension) in Notepad. Copy the whole thing (including -----BEGIN OPENSSH PRIVATE KEY-----).
   - Go to **GitHub Repo > Settings > Secrets > Actions**.
   - Create these secrets:
     - SERVER_HOST: Your Ubuntu Server IP (e.g., 123.45.67.89)
     - SERVER_USER: Your username (e.g., root or ubuntu)
     - SSH_PRIVATE_KEY: Paste the private key content here.

Step 2: Update the Workflow File
Modify your .github/workflows/docker-publish.yml file. We will add a second job called deploy that runs **only after** the build finishes successfully.

```
name: Build, Push, and Deploy

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

jobs:
  # JOB 1: BUILD AND PUSH (Same as before)
  build-and-push:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v4

      - name: Login to Docker Hub
        uses: docker/login-action@v3
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Build and Push App
        uses: docker/build-push-action@v5
        with:
          context: .
          file: ./Dockerfile
          push: true
          tags: ${{ secrets.DOCKER_USERNAME }}/livt-app:latest
          cache-from: type=gha
          cache-to: type=gha,mode=max

      - name: Build and Push Nginx
        uses: docker/build-push-action@v5
        with:
          context: .
          file: ./docker/nginx/Dockerfile
          push: true
          tags: ${{ secrets.DOCKER_USERNAME }}/livt-nginx:latest
          cache-from: type=gha
          cache-to: type=gha,mode=max

  # JOB 2: DEPLOY TO UBUNTU (New!)
  deploy:
    needs: build-and-push  # Wait for build to finish
    runs-on: ubuntu-latest

    steps:
      - name: Deploy to Ubuntu Server
        uses: appleboy/ssh-action@v1.0.3
        with:
          host: ${{ secrets.SERVER_HOST }}
          username: ${{ secrets.SERVER_USER }}
          key: ${{ secrets.SSH_PRIVATE_KEY }}
          script: |
            # 1. Go to your project folder
            cd /root/livt-project

            # 2. Pull the latest images we just pushed
            docker compose pull
    # 3. Restart containers with the new images
    docker compose up -d

    # 4. Run migrations (Optional but recommended)
    docker compose exec -T app php artisan migrate --force

    # 5. Clean up unused images to save disk space
    docker image prune -f
```

How it works

1. **needs: build-and-push**: This line ensures the deployment never happens if the build fails.
2. **appleboy/ssh-action**: This plugin creates a secure SSH tunnel using the key you stored in Secrets.
3. **The Script**: It executes the exact commands you would run manually.
   - **-T in exec**: We use exec -T because GitHub Actions is a non-interactive terminal (no TTY), preventing errors during the migration command.
   - **--force in migrate**: In production, Laravel asks "Are you sure?" before migrating. The --force flag answers "Yes" automatically.

Push change to GitHub. Watch the "Actions" tab. Once it finishes, check your live website—it should have your latest changes automatically.