WebPage & deployment

Introdução Engenharia Informática

Mário Antunes

November 17, 2025

Exercises

Practice Guide: Build & Deploy a Static Webpage

This guide will walk you through the entire process of creating a simple, static website, starting from a single HTML file and progressively building it up. We will finish by serving this website from a professional-grade Nginx web server running inside a Docker container.

Step 0: Install Required Software (Debian)

We need to set up our Debian (or Debian-based, like Ubuntu) system. We need a text editor, Docker, and its 'compose' plugin.

1. **Update Package Lists:** Open a terminal and run:

   ```
   sudo apt update; sudo apt full-upgrade -y; \
   sudo apt autoremove -y; sudo apt autoclean
   ```

2. **Install a Text Editor:** We recommend Visual Studio Code for its features. Students using WSL can install VSCode in Windows and connect it into the VM.

   ```
   # Install prerequisites
   sudo apt install curl wget gpg
   wget -qO- https://packages.microsoft.com/keys/microsoft.asc \
   | gpg --dearmor > microsoft.gpg
   sudo install -D -o root -g root -m 644 microsoft.gpg \
   /usr/share/keyrings/microsoft.gpg
   rm -f microsoft.gpg
   ```

   Create a `/etc/apt/sources.list.d/vscode.sources` file with the following contents to add a reference to the upstream package repository:

   ```
   Types: deb
   URIs: https://packages.microsoft.com/repos/code
   Suites: stable
   Components: main
   Architectures: amd64,arm64,armhf
   Signed-By: /usr/share/keyrings/microsoft.gpg
   ```

   ```
   # Install VS Code
   sudo apt install apt-transport-https
   sudo apt update
   sudo apt install code # or code-insiders
   ```

The Exercises

Create a folder for your project.

```
mkdir my-portfolio
cd my-portfolio
```

If you installed VS Code, open this folder by typing `code .`

Step 1: Your First "Hello, World" Page

Create your first HTML file:

```
nano index.html
```

Inside the editor, type:

```
<h1>Hello, World!</h1>
```

Save the file (in `nano`, press `Ctrl+O`, `Enter`, then `Ctrl+X`). Find this `index.html` file in your file explorer, right-click it, and open it with Firefox. You'll see your text! The "URL" in your browser will be a local file path (e.g., `file:///home/your-user/my-portfolio/index.html`).

Step 2: Add a Real HTML5 Skeleton

A real HTML file needs a "skeleton." Replace the content of `index.html` with this:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My Portfolio</title>
</head>
<body>
    <h1>Hello, World!</h1>
</body>
</html>
```

- `<head>`: Contains "meta" information *about* the page.
- `<body>`: Contains the *visible content* of the page.
- `<meta charset="UTF-8">`: Ensures all text characters (like emojis) display correctly.
- `<meta name="viewport">`: Ensures your page looks good on mobile devices.

Save and refresh your browser. The page will look the same, but now your browser tab will say "My Portfolio".

Step 3: Add Semantic Structure

"Semantic" means "meaningful." We use HTML tags that describe their content. This is great for accessibility and SEO.

Replace the `<body>` section with this:

```
<body>
    <header>
        <h1>My Portfolio</h1>
        <nav>
            <a href="index.html">Home</a>
        </nav>
    </header>

    <main>
        <h2>Welcome</h2>
        <p>This is the main content of my website.</p>
    </main>

    <footer>
        <p>&copy; 2025 Your Name</p>
    </footer>
</body>
```

Refresh your browser. Now you have a header, a main content area, and a footer.

Step 4: Add "About Me" Content

Let's add a proper section to the page. Inside your `<main>` tag, right below the first `<p>` tag, add a new `<article>`:

```
    <main>
        <h2>Welcome</h2>
        <p>This is the main content of my website.</p>

        <!-- ADD THIS SECTION -->
        <article>
            <h3>About Me</h3>
            <p>I am a first-year student learning about web development. I am building th
            <p>In my spare time, I enjoy [Your Hobby 1] and [Your Hobby 2].</p>
        </article>
        <!-- END OF NEW SECTION -->

    </main>
```

Exercise: Change the placeholder text and fill in your own hobbies.

Step 5: Add a `mailto:` Link

A `mailto:` link will open the user's default email client. Let's add one to the footer.

Edit your `<footer>` section to look like this:

```
    <footer>
        <p>Contact me: <a href="mailto:your.email@example.com">your.email@example.com</a>
        <p>&copy; 2025 Your Name</p>
    </footer>
```

Save and refresh. Click the link to see it work.

Step 6: Link an External CSS File

Our HTML is getting full. It's time to separate our styles (CSS) into a new file.

1. Create a `css` folder:

   ```
   mkdir css
   ```

2. Create a new file for our styles:

   ```
   nano css/style.css
   ```

3. Add one test rule to `css/style.css`:

   ```
   body {
       background-color: #f0f0f0; /* A light gray background */
   }
   ```

4. **Link the CSS file.** Open `index.html` and add this line inside the `<head>` section:

   ```
   <head>
       <meta charset="UTF-8">
       <meta name="viewport" content="width=device-width, initial-scale=1.0">
       <title>My Portfolio</title>
       <link rel="stylesheet" href="css/style.css"> <!-- ADD THIS LINE -->
   </head>
   ```

Save both files and refresh your browser. Your page should now have a light gray background.

Step 7: Add Basic CSS (Fonts & Colors)

Now we can add more styles. Open `css/style.css` and add these rules:

```
/* This rule was already here */
body {
    background-color: #f0f0f0;

    /* ADD THESE */
    font-family: Arial, sans-serif;
    line-height: 1.6;
    color: #333;
}

header h1 {
    color: #004a8c; /* A nice dark blue */
}

a {
    color: #005a9c;
    text-decoration: none; /* Remove the underline */
}

a:hover {
    text-decoration: underline; /* Add underline back on hover */
}
```

Save and refresh. Your page will now have new fonts and colors.

Step 8: Add Layout Styling (Box Model)

Let's make our page look less "stuck" to the left side. We'll add some layout styles to our `css/style.css` file. This will center the content in "cards."

```
/* Add these new rules to the bottom of css/style.css */

header, main, footer {
    max-width: 800px; /* Stop the content from getting too wide */
    margin: 20px auto; /* 20px top/bottom, 'auto' left/right centers it */
    padding: 20px;
    background-color: #fff;
    border-radius: 8px; /* Rounded corners */
    box-shadow: 0 2px 5px rgba(0,0,0,0.1); /* A subtle shadow */
}

nav a {
    margin-right: 15px;
    font-weight: bold;
}

footer {
    text-align: center;
    font-size: 0.9em;
    color: #777;
}
```

Save and refresh. Your site now has a modern, centered, card-based layout.

Step 9: Add Media (Image)

An "About Me" section needs a photo.

1. Create an `images` folder:

   ```
   mkdir images
   ```

2. Find a profile picture, name it `profile.jpg`, and save it in the `images` folder. If you just want a placeholder, run this command:

```
wget "https://placehold.co/400×400/004a8c/fff?text=Me" -O images/profile.jpg
```

3. In `index.html`, find your "About Me" `<article>` and add the `<img>` tag:

```
<article>
    <h3>About Me</h3>
    <img src="images/profile.jpg" alt="A photo of me" width="150">
    <p>I am a first-year student ... </p>
    <p>In my spare time, I enjoy ... </p>
</article>
```

4. Let's make the image look nice. Add this to `css/style.css`:

```
article img {
    float: left; /* Make text wrap around the image */
    margin-right: 15px;
    border-radius: 50%; /* Make the image a circle */
    width: 150px;
    height: 150px;
}
```

Save and refresh. You now have a floating, circular profile picture.

Step 10: Add Media (Video)

Let's add a small video clip.

1. Create a `videos` folder:

```
mkdir videos
```

2. Let's download a small, free sample video:

```
wget "https://www.w3schools.com/html/mov_bbb.mp4" -O videos/my-clip.mp4
```

(This is a 10MB clip from "Big Buck Bunny," a common test video).

3. In `index.html`, add a new `<section>` *inside* your `<main>` tag:

```
<main>
    <h2>Welcome</h2>
    <!-- ... your <p> and <article> tags ... -->

    <!-- ADD THIS NEW SECTION -->
    <section>
        <h3>My Favorite Short Film</h3>
        <video controls width="100%"> <!-- 'width="100%"' makes it responsive -->
            <source src="videos/my-clip.mp4" type="video/mp4">
            Your browser does not support the video tag.
        </video>
    </section>

</main>
```

Save and refresh. Your page now has an embedded video player.

Step 11: Re-organize for Deployment

For a professional deployment, we need to separate our **source code** from our **deployment configuration**. We will put all our public website files into a single `src` folder.

1. Create the `src` folder:

```
mkdir src
```

2. Move all your website files into it:

```
mv index.html src/
mv css src/
mv images src/
mv videos src/
```

Your project structure should now look like this:

```
my-portfolio/
├── src/
│   ├── index.html
│   ├── css/
│   │   └── style.css
│   ├── images/
│   │   └── profile.jpg
│   └── videos/
│       └── my-clip.mp4
```

Important: Go back to your browser and refresh. **Your page is now broken!** Why? Because the `file:///` path is wrong. This is fine! We are about to fix this for good by running a real server.

Step 12: Deploy with Docker and Nginx

This is the final step. We will serve our `src` folder using a real Nginx web server.

1. In the root of your `my-portfolio` folder (the one *containing* the `src` folder), create a new file named `docker-compose.yml`:

   ```
   nano docker-compose.yml
   ```

2. Add this "recipe" to the file:

   ```
   services:
     webserver:
       image: nginx:alpine
       container_name: my_portfolio_site
       ports:
         - "8080:80"
       volumes:
         - ./src:/usr/share/nginx/html
   ```

   What this means:

   - `image: nginx:alpine`: Use the small, official Nginx web server.
   - `ports: - "8080:80"`: Map port **8080** on our computer to port **80** in the container.
   - `volumes: - ./src:/usr/share/nginx/html`: This is the magic. It tells Docker to take our `./src` folder and make it available *inside* the container at `/usr/share/nginx/html`, which is the exact folder Nginx serves files from.

3. **Run it!** Make sure you are in the `my-portfolio` directory (the one with `docker-compose.yml`). Run:

   ```
   docker compose up -d
   ```

   (`-d` means "detached," so it runs in the background).

4. **Visit Your Live Site!** Open your browser and go to:

   ```
   http://localhost:8080
   ```

You should see your complete, styled website, with images and video, being served by a professional-grade Nginx web server!

To stop the server, just run:

```
docker compose down
```

Congratulations!

You have successfully:

- Set up a Debian environment with modern development tools.
- Written a valid, semantic HTML5 webpage.
- Added text content, a `mailto:` link, images, and a video.
- Styled that page with an external, responsive CSS file.
- Packaged and deployed your website inside a Docker container using Nginx.

This is the fundamental workflow for modern static web development.