Day 20

Learned about CI/CD Pipelining and Docker

Creating a CircleCI/CD Pipeline:

1. Project Setup:

- Create a CircleCI account (if you don't have one).
- Connect your GitHub repository containing your website code to CircleCI.
- Create a .circleci/config.yml file in your repository's root directory.

2. Sample Configuration (config.yml):

YAML

```
version: 2.1  # Specify CircleCI version


jobs:
  build:  # Job name for building the website
    docker:  # Use a Docker container to ensure consistent environment
      - image: circleci/node:lts-browsers  # Pre-configured Node.js environment
    steps:
      - checkout  # Checkout your project code from the Git repository


      - restore_cache:  # Restore dependencies from cache for faster builds (optional)
          keys:
            - <dependency-cache-key>  # Replace with a unique key for your project


      - install:  # Install dependencies
          npm ci


      - run:  # Build website commands
          npm run build  # Replace with your build command (e.g., `gulp build`)


      - save_cache:  # Cache dependencies for faster future builds (optional)
          paths:
            - node_modules
          key: <dependency-cache-key>  # Same key as restore_cache
  test:  # Job name for running tests (if applicable)
    # ... add test commands here ...

  deploy:  # Job name for deployment (optional)
    # ... add deployment commands here ...

workflows:
  build-and-test:  # Workflow to run build and test jobs sequentially
    jobs:
      - build
      - test:  # Requires the 'test' job to be defined above
```

Use code with caution.

content_copy

3. Customize the Configuration:

- Replace placeholders like <dependency-cache-key> with appropriate values.
- Adapt npm run build to your specific build command.
- If you have tests, add a test job with test running commands.
- For deployment, implement a deploy job with deployment-specific commands (consider factors like deployment environment, secrets management, etc.).
- CircleCI offers extensive documentation and configuration options for various languages, frameworks, and deployment targets. Refer to their documentation for more advanced use cases: https://circleci.com/docs/

Leveraging Technologies:

While the provided technologies don't directly require special CircleCI configuration, consider these points:

- The Docker container ensures a consistent environment, especially for libraries like parallax.js or wow.js that rely on JavaScript versions or specific dependencies.
- Use build commands appropriate for your project's build process (e.g., gulp build if you use a build tool like Gulp).
- If you use Google Tag Manager, deployment might involve configuration file management or interaction with a tag management API.
- For SEO and error handling (robots.txt, sitemaps, 404.html), deployment might involve copying or generating these files as part of the deployment process.

Additional Considerations:

- **Empty Websites:** For empty websites, you might not need a complex build process. The CircleCI pipeline might simply involve

Docker is a platform designed to help developers build, ship, and run applications quickly. It achieves this by using a concept called containers.

Here's a breakdown of Docker and its key components:

Containers:

- **What are they?** Containers are lightweight, self-contained units of software that package code and all its dependencies (libraries, system tools, settings) for a particular application. They share the operating system kernel with other containers on the same machine, but they run in isolation, ensuring consistency and avoiding conflicts.
- **Benefits:**
  - **Portability:** Containers run consistently across different environments (Windows, macOS, Linux) because they bundle everything needed.
  - **Isolation:** Containers share resources but run in isolation, preventing conflicts between applications.
  - **Efficiency:** Containers use fewer resources than virtual machines, making them lightweight and fast to start.
  - **Scalability:** You can easily scale applications by starting more containers.

Docker Engine:

- Docker Engine is the core software that manages containers. It includes:
  - **dockerd:** The daemon process that runs in the background and manages container creation, deletion, and execution.
  - **docker cli:** The command-line interface you use to interact with Docker Engine (e.g., building, running, stopping containers).

Docker Hub:

- Docker Hub is a public registry where you can find and share Docker images. It's like an app store for containers, containing millions of pre-built images for various applications, libraries, and tools. You can pull existing images from Docker Hub or push your own custom images to share.

Building Your Own Images:

- You can create custom Docker images using Dockerfiles. A Dockerfile is a text document that specifies the instructions for building a Docker image. It defines the base image, installs dependencies, copies your application code, and configures the environment.

Benefits of Using Docker:

- **Faster Development:** Docker simplifies development by providing a consistent environment and making it easy to share code with others.
- **Efficient Deployment:** Containers are lightweight and portable, making them ideal for deploying applications to production environments.

- **Scalability:** Docker makes it easy to scale applications by starting more containers.
- **Microservices Architecture:** Docker is well-suited for building microservices architectures, where applications are broken down into smaller, independent services.

Getting Started with Docker:

There are several ways to get started with Docker:

- **Docker Desktop:** A free, user-friendly application for Windows, macOS, and Linux that lets you build, run, and manage containers with a graphical interface.
- **Docker Hub:** Explore pre-built images and gain insights into using Docker.
- **Docker documentation:** Provides comprehensive guides and tutorials to learn Docker concepts and commands in detail.