Below is a **clear, detailed, and beginner-friendly explanation** of **Docker**, **Kubernetes**, **CI**, and **CD**, including what they are and how they are used—along with real-world examples.

---

1. Docker – What It Is & Why It's Used

What is Docker?

Docker is a platform that allows developers to **package applications with all their dependencies into containers**.

A **container** is a lightweight, isolated environment that contains:

- The application code
- Libraries
- System tools
- Configuration
- Dependencies

This makes the application run **the same way everywhere**—on any computer, server, or cloud platform.

Why use Docker?

1. Consistency

Eliminates "works on my machine" issues.

2. Portability

A container runs the same on:

- Windows
- Linux
- macOS
- Cloud servers (AWS, Azure, GCP)

3. Speed

Containers start in **seconds**, much faster than virtual machines.

□ 4. Isolation

Each container runs independently—no conflicts.

□ 5. Scalability

Easy to scale applications by running multiple container instances.

□ Real-world example of Docker

Imagine a Python app that needs:

- Python 3.10
- Flask
- Redis

Package everything into a **Docker image**.

Anyone who runs that image gets the same environment.

□ 2. Kubernetes – What It Is & Why It's Used

□ What is Kubernetes (K8s)?

Kubernetes is an **orchestration system** for managing containers (usually Docker containers) at scale.

Think of Docker as a *shipping container*
and Kubernetes as the *port that organizes thousands of containers*.

□ What does Kubernetes do?

□ 1. Deploys containers automatically

You define how many copies (replicas) of your app you want.

□ 2. Self-healing

If a container crashes, Kubernetes restarts it.

□ 3. Load balancing

Distributes traffic evenly across containers.

□ 4. Auto-scaling

Adds or removes containers based on CPU or traffic.

□ 5. Service discovery

Helps containers find and talk to each other.

□ 6. Rolling updates

Updates the app without downtime.

□ Real-world example of Kubernetes

A large company (Netflix, Amazon, TikTok) may run **thousands of containers**.

Kubernetes ensures:

- Apps stay running
- New versions deploy smoothly
- Traffic is balanced
- Containers scale up during high usage

□ 3. CI – Continuous Integration

□ What is Continuous Integration (CI)?

CI is the practice of automatically:

- Building code
- Running tests
- Checking quality
  whenever developers push code to a shared repository.

□ Why CI is important?

□ 1. Finds bugs early

Tests run every time code changes.

□ 2. Ensures code quality

Linting, formatting, and security scans run automatically.

□ 3. Automates builds

No manual steps to compile or prepare code.

□ 4. Speeds up development

Developers get immediate feedback.

□ Common CI tools

- GitHub Actions
- GitLab CI
- Jenkins
- CircleCI
- Azure DevOps

□ Example of CI

Developer pushes code → CI pipeline:

1. Install dependencies
2. Run tests
3. Build Docker image
4. Notify developer of results

□ 4. CD – Continuous Delivery / Continuous Deployment

There are **two forms**:

□ (A) Continuous Delivery

- Code is automatically **tested and prepared for release**
- A human must approve deployment

Example:
"Click Deploy to Production"

□ (B) Continuous Deployment

- Code is automatically **released to production** after passing tests
- No manual approval needed

□ Why CD is useful?

□ 1. Faster releases

Deploy many times a day.

□ 2. Reduced human error

Automation eliminates mistakes.

□ 3. Safe rollbacks

If something breaks, automatically revert.

□ Common CD tools

- Argo CD (for Kubernetes)
- Tekton
- Spinnaker
- GitHub Actions CD
- GitLab CD
- Jenkins X

□ Example of CD

1. Code pushed to GitHub
2. CI runs tests & builds Docker image
3. CD deploys image to Kubernetes cluster
4. Kubernetes rolls out new version without downtime

□ How Docker, Kubernetes, CI, CD work together

Here is the full modern DevOps pipeline:

1 □ **Developer writes code**
2 □ Code pushed to GitHub (or GitLab)
3 □ **CI pipeline runs**

- Build Docker image
- Run tests
- Security scans

4 □ **CD pipeline deploys automatically**

- Push Docker image to container registry
- Update Kubernetes deployments

5 □ **Kubernetes handles the rest**

- Load balancing
- Auto-scaling

- Self-healing
- Rolling updates

□ Summary in One Line

- **Docker** → packages applications into containers
- **Kubernetes** → runs and manages containers at scale
- **CI** → tests & builds your code automatically
- **CD** → deploys your code automatically

Below is a **step-by-step beginner-friendly guide** on how to use **Docker** and **Kubernetes (via Minikube)** on a **Windows PC** to run and manage your **Bootstrap 5 web pages**.

Your use case is simple (static web pages), so I'll show you:

- How to containerize your Bootstrap site with Docker
- How to run it on Windows Docker Desktop
- How to deploy it to Kubernetes using Minikube or Docker Desktop Kubernetes

---

□ 1. Install Docker on Windows

□ Requirements

You need:

- **Windows 10/11 64-bit**
- **WSL 2 enabled** (Docker Desktop will set it up if not present)

□ Installation Steps

1. Download Docker Desktop
   □ Search for: *Docker Desktop for Windows*
   (Or from Docker's official site)
2. Install normally
3. When asked:
   - ✔ Enable WSL 2 backend
   - ✔ Enable Kubernetes (optional, if you want to use the built-in Kubernetes)
4. Restart PC after installation
5. Verify Docker is running:
   Open PowerShell or Terminal:
6. `docker --version`

□ 2. Prepare Your Bootstrap 5 Website

Create a folder:

```
my-bootstrap-site/
    index.html
    about.html
    css/
    js/
    assets/
```

You can use any static Bootstrap pages.

□ 3. Create a Dockerfile (VERY EASY)

Inside the folder, create a file named **Dockerfile**:

```
FROM nginx:alpine

# Remove default NGINX static files
RUN rm -rf /usr/share/nginx/html/*

# Copy your Bootstrap website files
COPY . /usr/share/nginx/html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

This uses a lightweight **NGINX server** to host your Bootstrap site.

□ 4. Build the Docker Image

In the project folder, run:

```
docker build -t bootstrap-site .
```

Then run it:

```
docker run -d -p 8080:80 bootstrap-site
```

Now open:

□ http://localhost:8080

Your Bootstrap 5 site is now running in Docker!

□ 5. OPTIONAL: Use Docker Compose

Create `docker-compose.yml`:

```
version: "3.9"
services:
  web:
    image: bootstrap-site
    build: .
    ports:
      - "8080:80"
```

Run:

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

□ 6. Enable Kubernetes on Windows

Option A — Use Kubernetes from Docker Desktop (Easiest)

Go to:
Docker Desktop > Settings > Kubernetes > Enable Kubernetes

Wait for it to start (first time takes 5–10 mins).

Option B — Install Minikube (Most common for learning)

1. Install **kubectl**:
2. `winget install Kubernetes.kubectl`
3. Install Minikube:
4. `winget install minikube`
5. Start Minikube:
6. `minikube start --driver=docker`

---

7. Create Kubernetes Deployment and Service

Inside your project folder, create:

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bootstrap-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: bootstrap
  template:
    metadata:
      labels:
        app: bootstrap
    spec:
      containers:
      - name: bootstrap-container
        image: bootstrap-site
        imagePullPolicy: Never
        ports:
        - containerPort: 80
```

imagePullPolicy: Never tells Kubernetes to use the local Docker image.

---

service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: bootstrap-service
```

```
spec:
  type: NodePort
  selector:
    app: bootstrap
  ports:
  - port: 80
    targetPort: 80
    nodePort: 30080
```

□ 8. Deploy to Kubernetes

Apply the YAML files:

```
kubectl apply -f deployment.yaml
kubectl apply -f service.yaml
```

Check everything:

```
kubectl get pods
kubectl get svc
```

□ 9. Access the Bootstrap Site in Kubernetes

Minikube:

```
minikube service bootstrap-service
```

Docker Desktop Kubernetes:

Open browser:

□ http://localhost:30080

Your **Bootstrap site is now running inside Kubernetes**!

□ 10. Summary Workflow

1. Write Bootstrap site
2. Create Dockerfile
3. Build Docker image
4. Run in Docker

5. Enable Kubernetes
6. Deploy using deployment + service YAML
7. Access via NodePort