# DevOps Practices and Tools: A Beginner's Guide

## 1. Git and GitHub: Version Control and Collaboration

### Overview

- Git is a distributed version control system for tracking changes in source code.  
- GitHub is a platform for hosting Git repositories, enabling collaboration and integration with CI/CD tools.

### Practical Example

- Scenario: A team of developers collaborates on a web application.  
 - Developers use Git to clone the repository, make changes, and commit them locally.  
 - GitHub is used to push these changes to a central repository, enabling pull requests for code review.

### Steps

1. Clone a repository:  
 git clone https://github.com/username/repository.git  
2. Add and commit changes:  
 git add .  
 git commit -m "Description of changes"  
 git push origin branch-name  
3. Use GitHub to open a pull request for code review.

## 2. GitHub Actions: Continuous Integration

### Overview

- Automates tasks like building, testing, and deploying applications on every push or pull request.  
- Provides an easy-to-configure workflow system directly within GitHub.

### Practical Example

- Scenario: Automate testing for a Node.js application.

### Workflow

1. Developers push code changes to GitHub.  
2. GitHub Actions runs a pre-configured workflow to test the application automatically.

### Steps

1. Add a GitHub Actions YAML file (`.github/workflows/ci.yml`):  
 name: Node.js CI  
  
 on: [push, pull\_request]  
  
 jobs:  
 build:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v3  
 - uses: actions/setup-node@v3  
 with:  
 node-version: '16'  
 - run: npm install  
 - run: npm test

## 3. Docker: Containerization

### Overview

- Docker packages an application and its dependencies into a portable container.  
- Containers ensure consistent environments across development, testing, and production.

### Practical Example

- Scenario: Deploy a Python application with its dependencies using Docker.

### Workflow

1. Create a `Dockerfile` to define the container image.  
2. Build and run the image locally or in a cloud environment.

### Steps

1. Create a `Dockerfile`:  
 FROM python:3.9-slim  
  
 WORKDIR /app  
  
 COPY requirements.txt ./  
 RUN pip install -r requirements.txt  
  
 COPY . .  
  
 CMD ["python", "app.py"]  
2. Build and run the container:  
 docker build -t python-app .  
 docker run -p 5000:5000 python-app

## 4. Kubernetes: Orchestration

### Overview

- Kubernetes automates deployment, scaling, and management of containerized applications.  
- Key components include pods, deployments, and services.

### Practical Example

- Scenario: Deploy a web application on a Kubernetes cluster.

### Workflow

1. Define deployment and service YAML files.  
2. Apply configurations to the cluster using `kubectl`.

### Steps

1. Create a deployment YAML (`deployment.yaml`):  
 apiVersion: apps/v1  
 kind: Deployment  
 metadata:  
 name: web-app  
 spec:  
 replicas: 2  
 selector:  
 matchLabels:  
 app: web-app  
 template:  
 metadata:  
 labels:  
 app: web-app  
 spec:  
 containers:  
 - name: web-app  
 image: web-app:latest  
 ports:  
 - containerPort: 80  
2. Apply the deployment:  
 kubectl apply -f deployment.yaml